---

title: "What is Adapty Plugin for FlutterFlow" description: ""

metadataTitle: ""

import Zoom from 'react-medium-image-zoom'; import 'react-medium-image-zoom/dist/styles.css'; import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty is a versatile in-app purchase platform designed to help you grow your subscriber base. Whether youâ€™re just starting out or already have millions of users, Adapty simplifies setting up the best subscription prices, testing different strategies, and finding what works best for your appâ€™s success.

- **Subscriptions and In-App Purchases**: Adapty handles server-side receipt validation for you and syncs your customers across all platforms, including the web.
- **A/B Testing for Subscription Plans**: Test different prices, durations, trial periods, and visual elements to optimize your subscription offerings.
- **Powerful Analytics**: Access detailed metrics to better understand and improve your appâ€™s monetization.
- **Integrations**: Adapty seamlessly connects with third-party analytics tools like Amplitude, AppsFlyer, Adjust, Branch, Mixpanel, Facebook Ads, AppMetrica, custom Webhooks, and more.

The Adapty plugin for FlutterFlow allows you to harness all of Adaptyâ€™s features without writing a single line of code. Design paywall pages in FlutterFlow, load Adapty paywalls, enable purchases, release your app, and voilÃ â€”youâ€™ll see detailed analytics of customer purchases directly in the Adapty Dashboard.

Get started with Adapty on FlutterFlow and enjoy 3 months for free!

---

title: "Getting started" description: ""

metadataTitle: ""

import Zoom from 'react-medium-image-zoom'; import 'react-medium-image-zoom/dist/styles.css'; import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

To start using all Adapty features in FlutterFlow, add the Adapty Plugin as a dependency in your FlutterFlow project:

1. In your project, click **Settings and Integrations** ifrom the left menu. img/mainsettings.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
2. In the **App Settings** section on the left, select **Project dependencies**. img/settingsdependencies.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
3. In the **Project Dependencies** window, click the **Add Dependency** button.

img/add-dependency.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} /> 4. From the list, choose **Adapty FF Plugun**. img/select*adapty*plugin.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

4. Click the **Add** button.

The Adapty FF Plugin will now be added as a dependency to your project. In the **Adapty FF Plugin** window, youâ€™ll find all the Adapty resources that have been imported into your project.

---

title: "Fetch Adapty paywall in FlutterFlow" description: ""

metadataTitle: ""

import Zoom from 'react-medium-image-zoom'; import 'react-medium-image-zoom/dist/styles.css';

We assume you'll design the paywall UI yourself using FlutterFlow's rich features. In parallel, you'll need to create the same paywall iin the Adapty Dashboard. You don't need to design the UI in Adapty, but be sure to add the productsâ€”subscriptions or non-subscriptionsâ€”that you want to sell on the paywall. Ensure that the paywall you design in FlutterFlow matches the number of products in the Adapty paywall.

Once you've created a paywall in the Adapty Dashboard, please create a placement and add your paywall to it. Adapty's placement system allows you to experiment with different paywalls, swapping one out for another over time without needing to release a new app version.

The only thing you need to hardcode in your mobile app is the placement ID. This means you hardcode the placement ID, and through it, you get the set of products configured for the specific paywall added to this placement.

Before proceeding, make sure you've done the following in the Adapty Dashboard::

1. Created at least one paywall and added at least one product to it.
2. Created at last one placement and added your paywall to it.

After completing these steps, you'll be able to call your paywall in your FlutterFlow app.

Step 1. Create page state variable

When setting up a paywall in your mobile app, you'll need a place to track the paywall data while the user is on that page. This is where a page state variable comes in. Think of it as a temporary storage box for paywall data specific to a single screen in your app.

Why Do You Need One?

The page state variable serves several key functions:

- **Stores Your Paywall**: It holds the paywall element when itâ€™s received.
- **Handles Errors**: If something goes wrong while fetching the paywall, this variable helps catch and handle the error.
- **Updates the Screen**: Once the paywall data is received, the screen can update with the list of products.

How to create a page state variable

1. On the page that is shown before the page with the paywall, click the **State Manageemnt** button in the right pane.

img/createpage*state*variable.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

2. In the **Local Page State Variables** pane that opens, click the **Add field** button.

img/addfield.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

3. In the **Field name** field, enter a clear name for your variable, like `paywall.

img/fieldname.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} /> 4. In the **Type** list, choose **Data Type**.

img/variabletype.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

5. In the second **Type** field, choose **AdaptyPaywall**.

img/adaptypaywall.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

6. Click the **Confirm** button.

Youâ€™ve now created a page state variable for the paywall. Next, we'll define how this paywall can be managed and how it should behave.

Step 2. Create Action Block

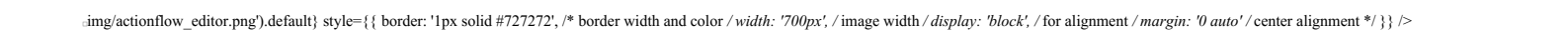
While you can add actions separately to a page, itâ€™s more convenient to create an action block for the paywall that defines how to retrieve it, process it, and handle any errors. You can then easily recall this action block whenever you need to re-fetch the paywall, such as in case of a connection failure. Weâ€™ll first prepare this action block and later use it in our app flow in Step 3.

Step 2.1. Open FlutterFlow Action Flow Editor

1. In the right pane, click the **Actions** button.

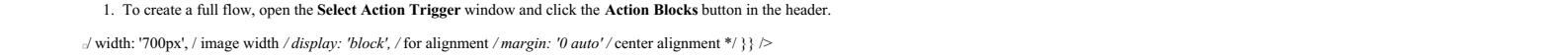
/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

2. n the opened pane, click **Open** button next to the **Action Flow Editor**.



Step 2.2. Start action block

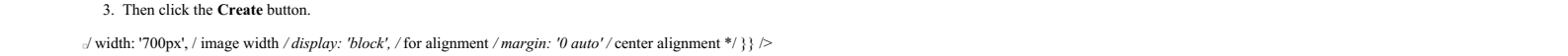
1. To create a full flow, open the **Select Action Trigger** window and click the **Action Blocks** button in the header.



2. In the **Action Blocks** window, toggle the **Page** toggle.

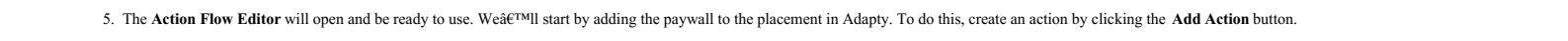
3. Then click the **Create** button.

3. Name your action block in the **Action Block Name** window. Use a name that makes sense to you; in this example, weâ€™ll use `getPaywallActionBlock.

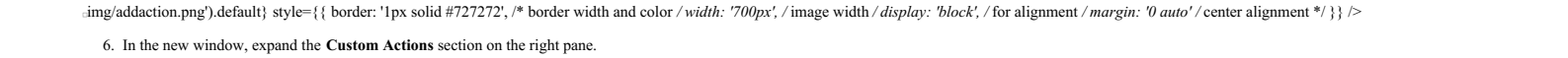


4. Click the **Create** button.
5. The **Action Flow Editor** will open and be ready to use. Weâ€™ll start by adding the paywall to the placement in Adapty. To do this, create an action by clicking the **Add Action** button.

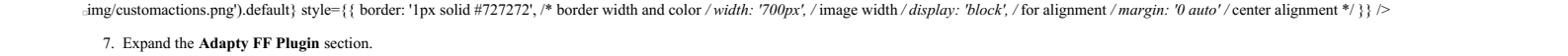
6. In the new window, expand the **Custom Actions** section on the right pane.



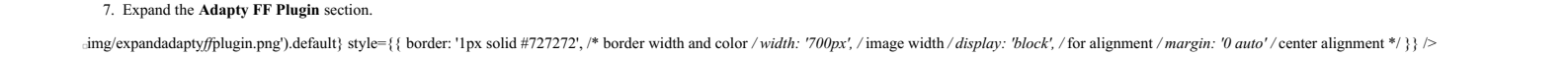
7. Expand the **Adapty FF Plugin** section.



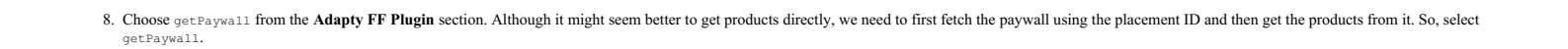
8. Choose `getPaywall` from the **Adapty FF Plugin** section. Although it might seem better to get products directly, we need to first fetch the paywall using the placement ID and then get the products from it. So, select `getPaywall`.



9. In the **Set Actions Arguments** section, enter the placement ID where you added the paywall in the **Value** field. In our example, the placement ID is `onboarding_placement`.



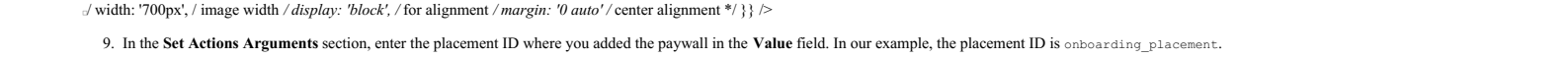
10. Set up the output of the paywall request by entering `paywallResult` (or a name of your choice) in the **Action Output Variable Name** field.



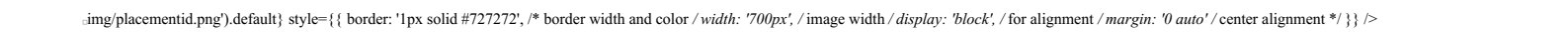
11. Now that the setup is complete, we can draw our action flow.

Step 2.3. Add condition to process successful Adapty paywal load and its failure

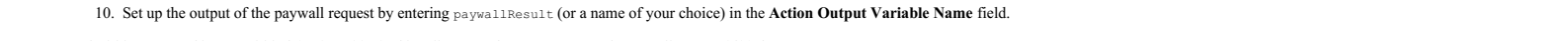
1. Click the **plus (+)** button below the **Custom Action** block.



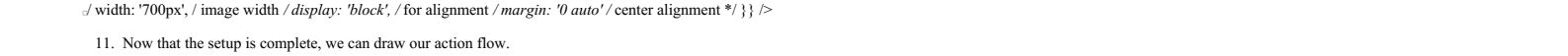
2. Since we need to handle both successful and failed paywall requests, add a condition by selecting **Add Conditional**.



3. In the **Action Output** section, select the action output variable created earlier (`paywallResult` in our example).



4. To verify the Adapty paywall was received successfully, check for the presence of a field with a value. In the **Available Options** list, choose **Has Field**.



5. In the **Field (AdaptyGetPaywallResult)** list, choose **value**.

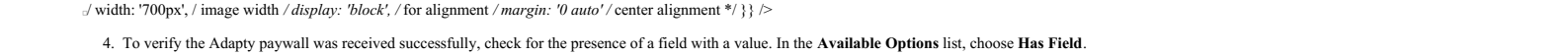
6. Click the **Confirm** button to finish setting up the condition.

Step 2.4. Process successful paywall receving case

1. Set up the action for when the paywall is successfully received by clicking the **plus (+)** button below the **TRUE** label.

2. Select **Add Action** from the list.

3. In the right pane, select **Update Page State** from the **State Management** section or use the search. width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
4. Define what should be updated on the page by clicking the **Add field** button.



5. In the **Search for field** window, click **paywall**. width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
6. Set the update type by choosing **Set Value** in the **Select Update Type** list.

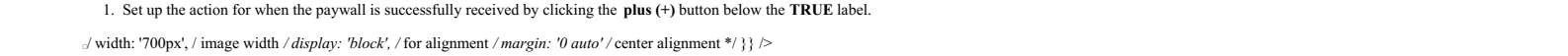
7. Click the `UNSET` value in the **Value** to Set field. In the **Set Variable** window, select the action output variable set in step 9 (`paywallResult).

8. In the **Available Options** field, choose **Data Structure Field**. width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
9. And finally, in the **Select Field** list, choose **value**.

10. Click the **Confirm** button to complete the setup. width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Step 2.5. Handle failed paywall retrieval

Configure what should happen if the paywall is not successfully received. For a complete list of error codes and how to handle them, refer to our AdaptyErrorCode class section.

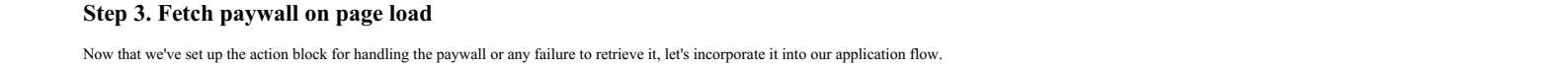


Once you've completed this step, close the FlutterFlow Action Flow Editor by clicking the **Close** button at the top-right corner of the **Action Flow Editor** window.

Step 3. Fetch paywall on page load

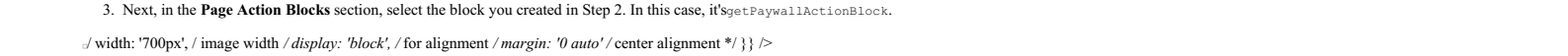
Now that we've set up the action block for handling the paywall or any failure to retrieve it, let's incorporate it into our application flow.

1. In the main window of your project, navigate to the page where the paywall should open, then click the **Add Action** button in the right pane.



::: info - If you're still in the Action Flow Editor, close it by clicking the **Close** button in the top-right corner, as shown in Step 2.5. - If you're in the main window and can't see the button, make sure you're in the **Page** widget in the left pane, and have opened the **Actions** pane in the right. :::

2. In the **Select Action** section on the right, choose the **On Page Load** option. You may need to scroll down to find it.



3. Next, in the **Page Action Blocks** section, select the block you created in Step 2. In this case, it's`getPaywallActionBlock`.

4. Enable the **Update Page After** toggle to display the paywall once it's retrieved.

Once you're done with this page, switch to the paywall page.

---

title: "Fetch paywall data in FlutterFlow" description: ""

metadataTitle: ""

import Zoom from 'react-medium-image-zoom'; import 'react-medium-image-zoom/dist/styles.css';

Now that youâ€™ve prepared everything to load the paywall, itâ€™s time to load paywall products.

Step1. Create a new page for a paywall

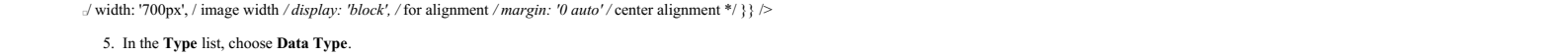
1. In the **Page Selector**, click the **Add Page, Component, or Flow** button.

2. Start with a blank page.
3. Name the page in the **New page** window. We will name it `paywallPage`.
4. Click the **Create page** button.

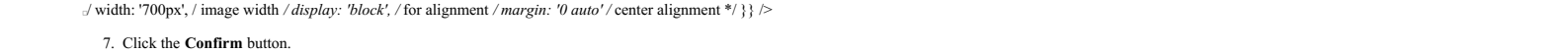
Step 2. Add a parameter for a paywall

1. Open the **Page Parameters** pane on the right.

3. In the pane, click the **Add Parameter** button.

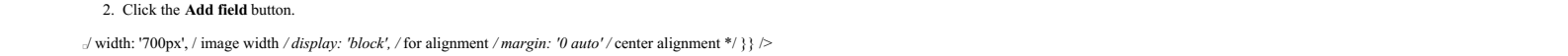


4. Name the parameter in the **Parameter Name** field. We will name it `paywall`.



5. In the **Type** list, choose **Data Type**.

6. In the second **Type** list, choose `AdaptyPaywall`.



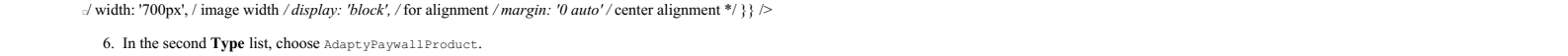
7. Click the **Confirm** button.

You've now added a parameter for an Adapty paywall.

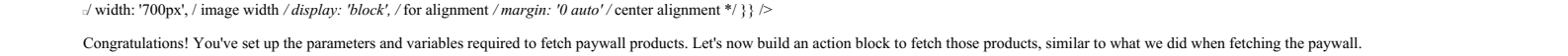
Step 3. Create product list variable

Now weâ€™ll create a variable to hold the products fetched with the paywall.

1. Switch to the **State Management** section in the right page.



2. Click the **Add field** button.

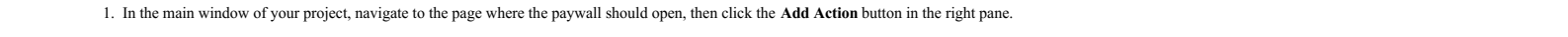


3. Name the variable in the **Field Name** field. We'll name it `products`.

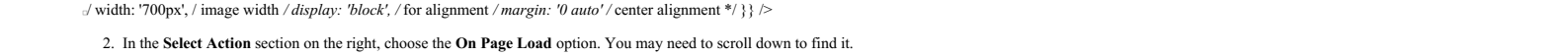
4. In the **Type** list, choose **Data Type**.

5. Select the **Is List** check-box.

6. In the second **Type** list, choose `AdaptyPaywallProduct`.



7. Click the **Confirm** button to save the changes.



Congratulations! You've set up the parameters and variables required to fetch paywall products. Let's now build an action block to fetch those products, similar to what we did when fetching the paywall.

Create action block to fetch paywall products

Youâ€™re already familiar with action blocks. Letâ€™s create one to fetch paywall products.

1. On the right, switch to the **Actions** section.

2. Open the Action Flow Editor.

3. In the **Select Action Trigger** window, click the **Action Blocks** button in the header.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

4. In the opened **Action Blocks** window, toggle **Page**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

5. Click the **Create** button.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

6. In the **Action Block Name** window, enter a name like **getPaywallActionBlock**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

7. Click the **Create** button.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

8. The **Action Flow Editor** will open. Configuring this action block to fetch products is similar to how we fetched the paywall. Start with the **Add Action** button.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

9. In the new window, expand the **Custom Actions** section on the right.

./img/customactions-products.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

10. Expand the **Adapty FF Plugin** section.

./img/expandadapty*ff*plugin.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

11. In the expanded **Adapty FF Plugin** section, choose **getPaywall**. Yes, we know, you'd prefer to get products all at once, but we need to go step by step: first, retrieve the paywall using the placement ID, and only then fetch the products. So, select **getPaywall**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

12. In the **Value** field in the **Set Actions Arguments** section, enter the ID of the placement that includes the paywall you want to display. In this example, the placement ID is `onboarding_placement`.

./img/placementid.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

13. After requesting the paywall, youâ€™ll need to capture its ID as a result. In the **Action Output Variable Name** field, type `paywallResult` (or any name that's meaningful to you).

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

14. With all the setup complete, youâ€™re ready to build the action flow.

Step 2.3. Add condition to process successful Adapty paywal load and its failure

1. Click the **plus (+)** button below the **Custom Action** block.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

2. We are going to process both ways: when the paywall is successfully received and when something went wrong and the paywall is not received. That is why we are going to add a condition - success of the paywall receiving. Choose **Add Conditional**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. In the **Action Output** section, find and click the action output variable we've created in step 9. In our examople, its `paywallResult`.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

4. Th easiest way to check that the Adapty paywall is successfully received is to check if it has a field with a value. Let;s do that. In the **Available Options** list, choose **Has Field**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

5. In the **Field (AdaptyGetPaywallResult)** list, choose **value**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

6. We've finished setting up the condition, click the **Confirm** button.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

---

title: "Presenting Adapty paywall in FlutterFlow" description: ""

metadataTitle: ""

import Zoom from 'react-medium-image-zoom'; import 'react-medium-image-zoom/dist/styles.css';

We assume that you will create the UI design on the paywall yourself by using the FlutterFlow rich functionality.

1. Open this paywall page. We will use a new page to avoid distracting your attention with bright pictures and multiple UI elements.

2. Open the **Page Parameters** pane in the right.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. In teh opened pane, click the **Add Parameter** button.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

4. ./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

---

title: "What is Adapty?" description: "Unlock the potential of your app's monetization strategy with Adapty â€“ a versatile in-app purchase platform designed to fuel subscriber growth. From startups to established giants, Adapty simplifies the process of optimizing subscription prices and testing various approaches to ensure your app's success" metadataTitle: "Discover Adapty: Your Ultimate In-App Purchase Platform"

slug: /

Adapty is a powerful and adaptable in-app purchase platform that helps you grow your subscriber base. Whether you're just starting or already have millions of users, Adapty makes it easy to set up the best subscription prices, test different approaches, and see what works best for your app's success.

Marketing automation and testing

- Subscriptions/in-app purchases SDK for iOS, Android, Flutter, React Native and Unity. Adapty performs server-side receipt validation for you and syncs your customers across all platforms, including web. It also works in Observer mode, so you can use SDK without changing your existing purchase infrastructure.
- A/B testing for subscription plans. Test different prices, durations, trial periods for your subscriptions as well as different visual elements.
- Analytics for the app economy. Detailed metrics related to your app monetization.
- Adapty can send subscription events to 3rd party analytics: Amplitude, AppsFlyer, Adjust, Branch, Mixpanel, Facebook Ads, AppMetrica, and custom Webhook.

Adapty works for developers, marketers, and executives

Marketers can directly engage users with promotional offers to return them to the service or upsell new products. With Adapty there is no need for programmers and analysts to manually extract segments.

For product managers and executives, Adapty has a dashboard with viable subscription metrics with daily/weekly/monthly reports to Slack and Emails.

---

title: "Quickstart guide" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Weâ€™re thrilled youâ€™ve decided to use Adapty! We want you to get the best results from the very first build. This guide will walk you through how to get started with Adapty

Creating a project and registering an application

Create a new application in Adapty that will represent the real application you want to manage in Adapty. To do so:

1. In the **App Settings** menu in the right-top corner of Adapty, open the General tab.
2. After the **General** tab opens, enter the name of the application, its category, and the reporting time zone.
3. You can configure other application settings. Mandatory settings are marked as **Required**.

Settings are saved automatically.

Configuring platforms

Configure SDKs to validate purchases and get subscription updates from one or both platforms in Adapty.

App Store configuration
In Adapty Dashboard, go to *App settings* > *iOS SDK* and fill in the fields using the instructions below.

In-app purchase API (StoreKit 2)

Upload in-app purchase keys to use Storekit 2 API. Please note that the fields will only be active when the app's Bundle ID is provided. Read how

Issuer ID

863f28c9-53f9-47eb-ac58-28d8e554494d

Key ID

K28NWK9SL1

Private key (.p8 file)

Click here or drag the file to this area to upload

App Store promotional offers

Upload subscription key to use Apple promotional offers. Read how

Subscription key ID

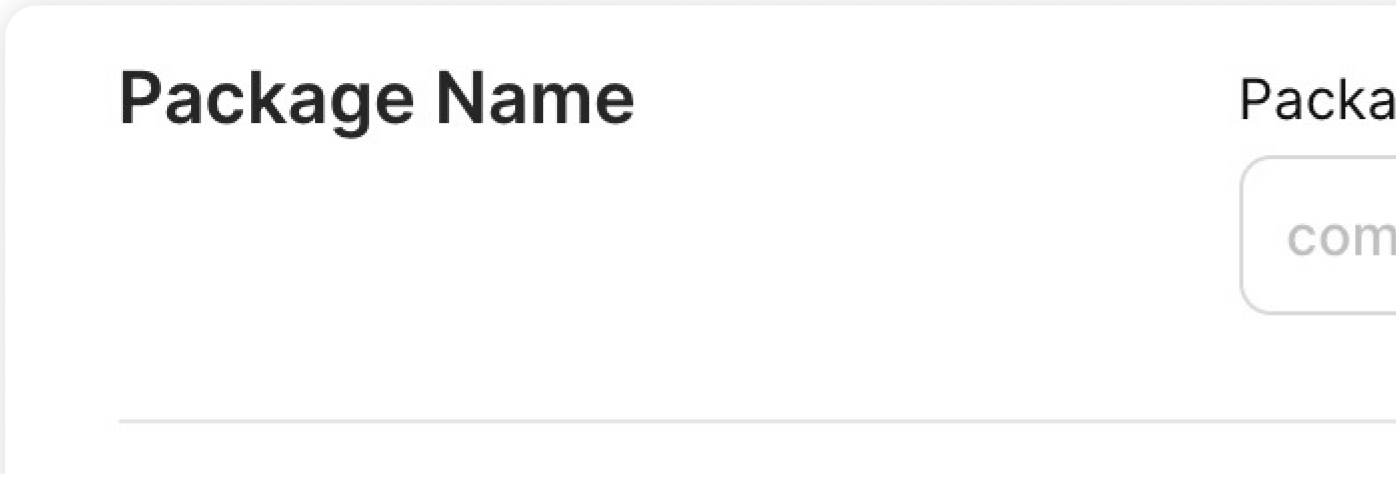
T2WMB822AB

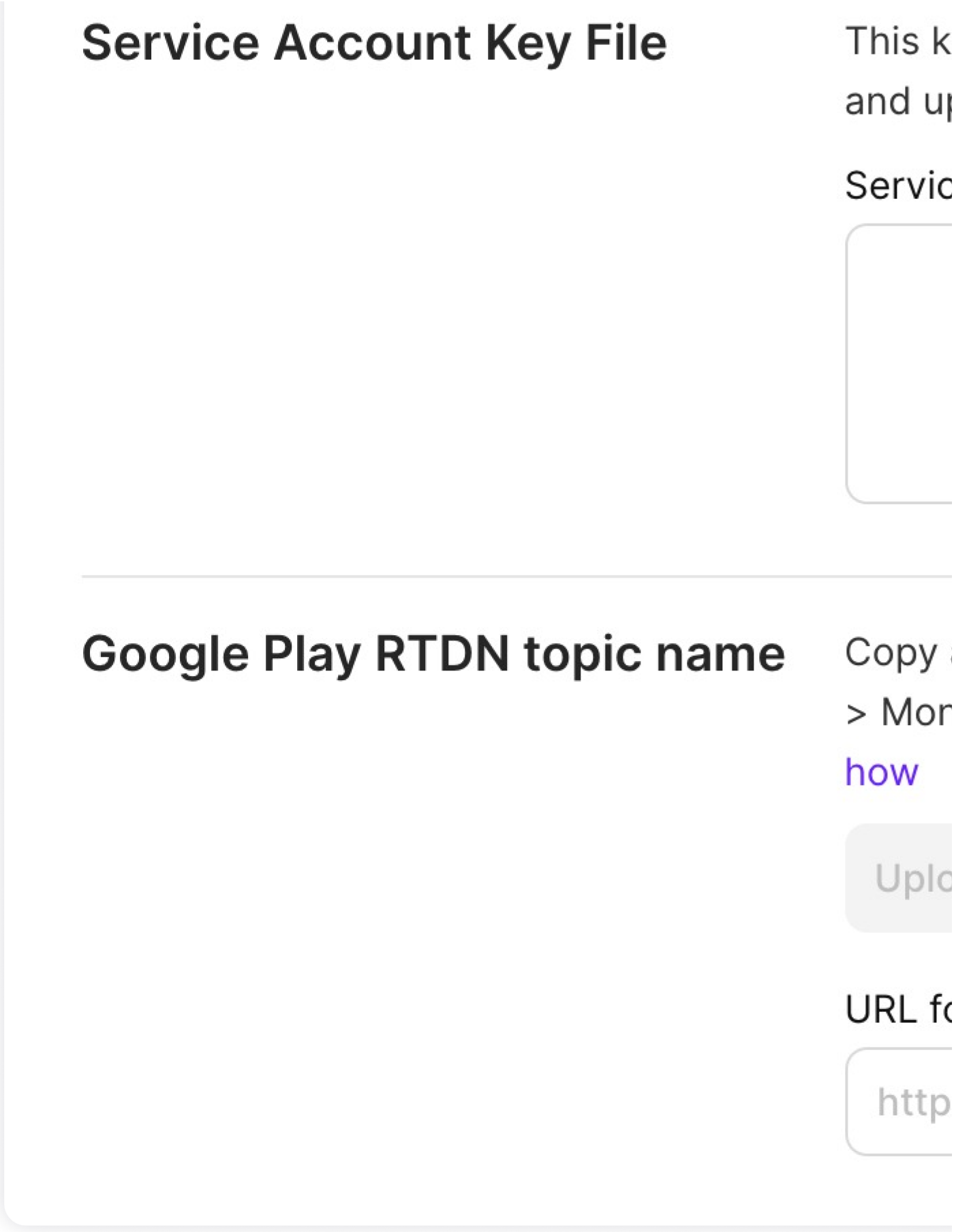
Subscription key (.p8 file)

Click here or drag the file to this area to upload

To find App Bundle ID, open App Store Connect, go to **My Apps**, and select the app whose ID you need. On the app page, in the drop-down **More** menu, select **About this App**. The app ID is displayed in the **Bundle ID** field.
â€“ How to connect to In-App Purchase API?
â€“ How to set URL for App Store Server Notifications? **â†'**

Play Store configuration
Go to *App settings* > *Android SDK* and fill in the fields.

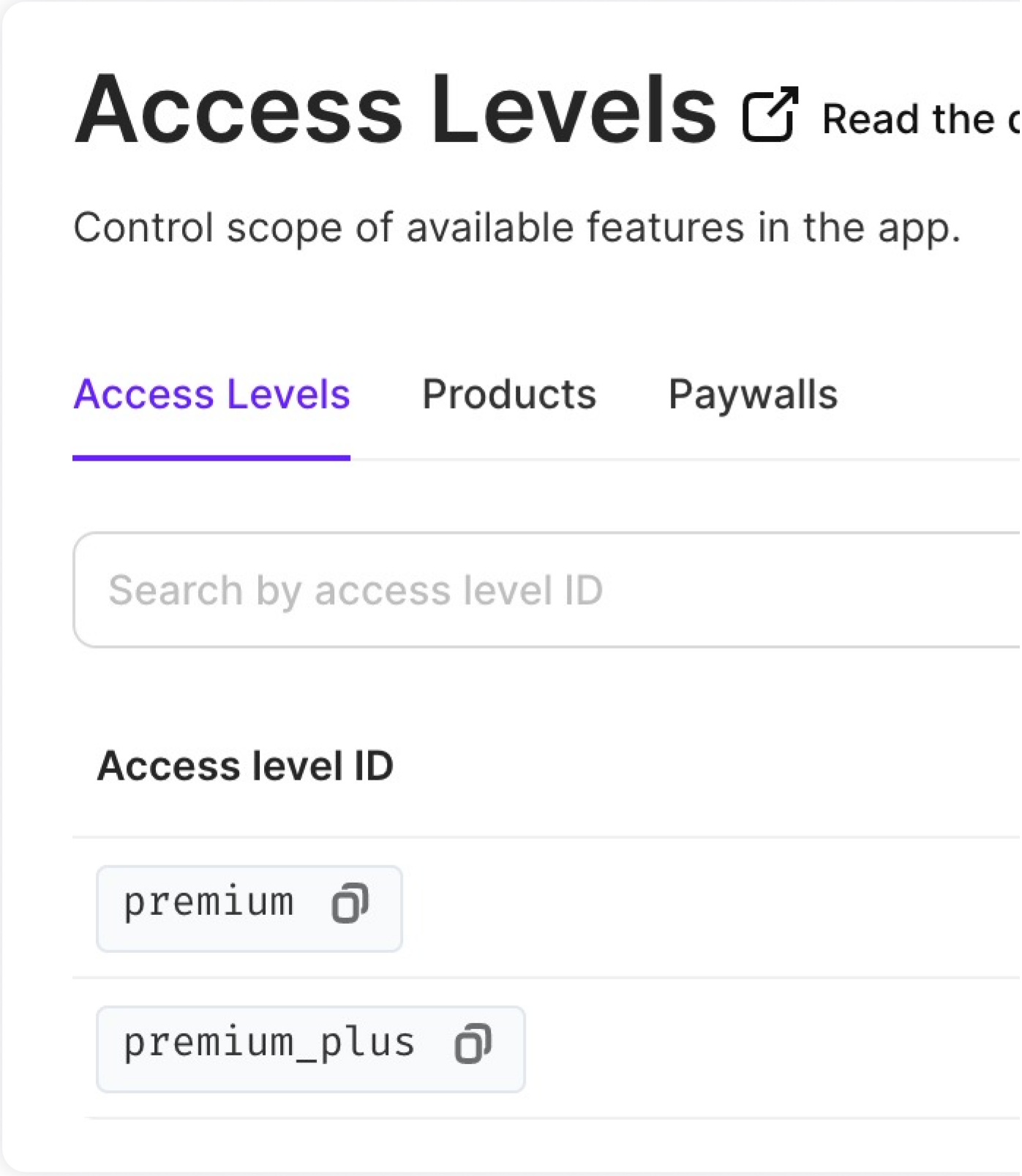




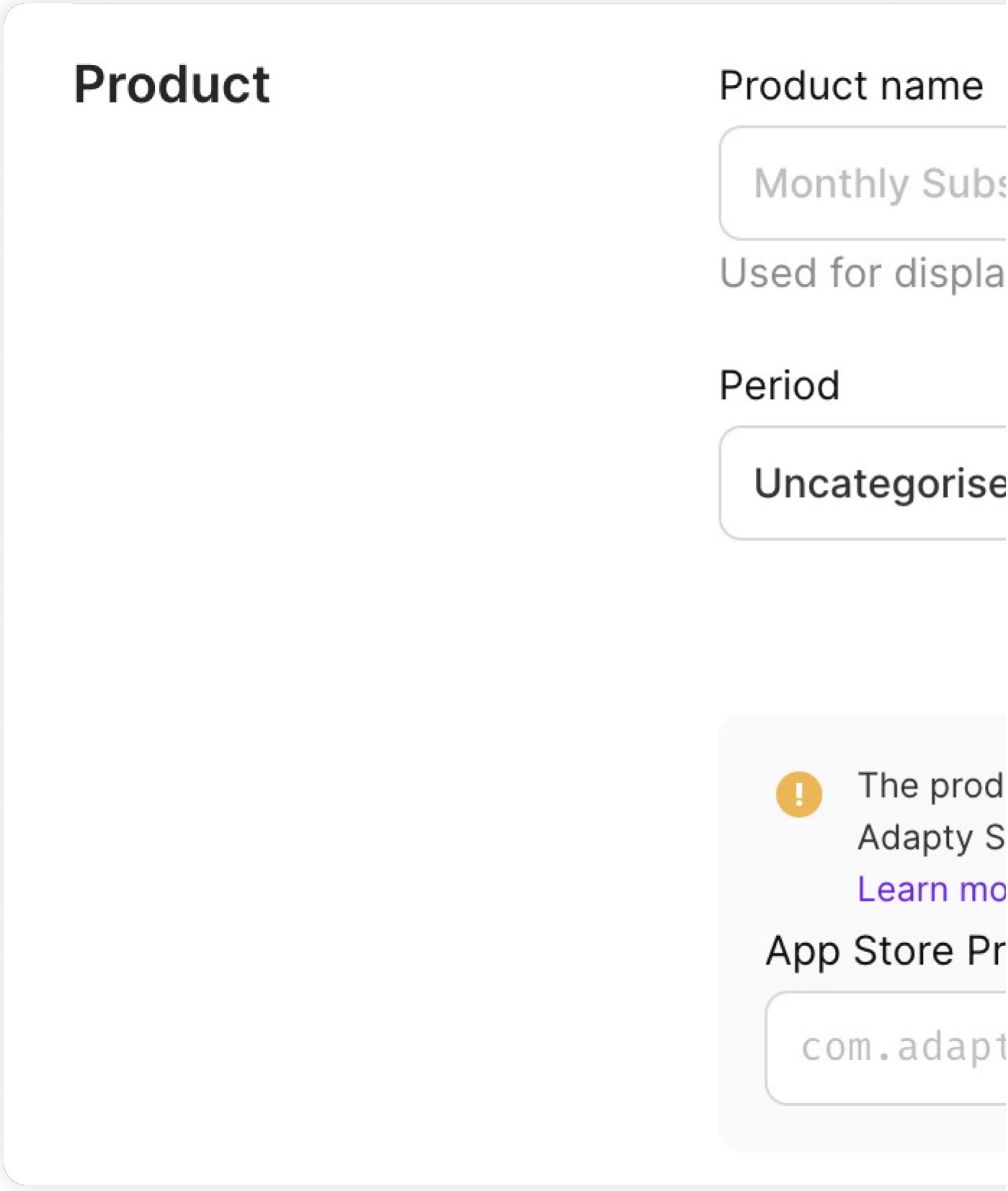
To find Package Name, open the Google Play Developer Console and select the app whose ID you need. The ID is specified next to the app's name and logo.
â€“ Where to find Service account key file? â†’
â€“ Where to find Real-time Developer Notifications (RTDN)? â†’

Creating a product

Add Access Level (optional)


Access Levels
Read the d
Control scope of available features in the app.
Access Levels
Products
Paywalls
Search by access level ID
Access level ID
premium
premium_plus

Add product



Setting up a paywall to show in the app

Create a paywall with this product

2023-07-03at_16.00.092x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Create a placement and add your paywall to it

2023-12-01at_17.21.382x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Installing Adapty SDK

Install and configure Adapty SDK in your app and be sure you have replaced the **"PUBLIC*SDK*KEY"** placeholder with your actual **Public SDK key**.

Bear in mind, that SDK calls must be made after calling `.activate()` method. Otherwise, we won't be able to authenticate requests and they will be canceled.

```
swift Adapty.activate("PUBLIC_SDK_KEY", customerUserId: "YOUR_USER kotlin override fun onCreate() { super.onCreate() Adapty.activate(applicationContext, "PUBLIC_SDK_KEY", customerUserId: "YOUR_USER_ID") } xml <dict> ... <key>AdaptyPublicSdkKey</key> <string>PUBLIC_SDK_KEY</string> </dict> xml <application ...> ... <meta-data android:name="AdaptyPublicSdkKey" android:value="PUBLIC_SDK_KEY" /> </application> ```typescript import { activateAdapty } from 'react-native-adapty';
```

const App: React.FC = () => { // ... useEffect(() => { activateAdapty({ sdkKey: 'PUBLIC*SDK*KEY' }); }, []); // ... } ```

Follow these guides for more info on:

- Displaying paywalls & products
- Setting up fallback paywalls

Configuring processing of purchases

Connecting Adapty to **In-App Purchase API** for iOS and adding both **package name** with **service account key file** for Android would be necessary to allow Adapty to successfully process purchasing events.

Subscription events

Here is what you can do to set up tracking of subscription events

| | | | :---------- | :---------- | | **For iOS** | **Update the App Store Server Notifications with our link** | | **For Android** | **Set up Real-time Developer Notifications (RTDN)** |

Integrations

Integrations with third-party analytics and attribution services require passing identifiers to the SDK.

| | | | :---------- | :---------- | | **.updateProfile()** | Use this to passing identifiers to Amplitude, Mixpanel, Facebook Ads, and AppMetrica | | **.updateAttribution()** | This method would be required for passing attribution data from AppsFlyer, Adjust, and Branch. Be sure to configure the integration of interest in Adapty Dashboard, by providing API key and event names |

Promo campaigns and promo offers

If you want to use Adapty along with Apple Promotional Offers, adding a subscription key will allow us to sign offers.

Notes

:::warning Don't forget about Privacy Labels

Learn more about the data Adapty collects and which flags you'd need to set for a review. :::

:::danger If you are using paywalls that were not built with Adapty Paywall Builder, make sure to send paywall views to Adapty using **.logShowPaywall()** method. Otherwise, paywall views will not be accounted for in the metrics and conversions will be irrelevant. :::

If you have any questions about integrating Adapty SDK, feel free to contact us using the website (we use Intercom in the bottom right corner) or just email us at support@adapty.io.

---

title: "Migrate to Adapty" description: ""

metadataTitle: ""

Migration has three steps:

1. Switching to Adapty SDK.
2. Changing Apple/ Google server2server notifications webhook.
3. (Optional) Importing historical data to Adapty to instantly pull statistics.

Let's quickly go through each part.

:::info Your subscribers will migrate automatically

All users who have ever activated subscription will move as soon as they open a new version with Adapty SDK. The subscription status validation and premium access will be restored automatically. :::

Installing Adapty SDK

Install Adapty SDK for your platform (iOS, Android, React Native, Flutter, Unity) in your app and replace your legacy logic with appropriate methods from Adapty SDK. Core things you need to replace:

- Checking an Access level to open a gated content;
- Making a purchase;
- Restoring purchase;
- Getting/setting information about your user.

:::tip Switching from another susbcription provider? Follow our guide for a detailed walk though giude: - Migration from RevenueCat (20 minutes) :::

Changing Apple server notifications

Apple and Google send us events that happen with users' subscriptions outside of the application (renewal, cancellation, pausing, refund, etc.) via App Store server notifications.

Adapty can work without this URL, but you'll get a limited feature set. For example, Integrations to 3rd party services will be delayed, subscription analytics won't be in real-time, and paywall A/B testing metrics won't be accurate.

When switching from a legacy system, sometimes you want two systems to work simultaneously for some time. In that case, you can use our raw events forwarding, where Adapty is a proxy server for your legacy system.

migrata.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Move historical data to Adapty

Moving historical data is optional and won't affect your subscribers' state. However, there are a number of reasons why it's better to do so:

1. **Analytics will work correctly instantly**. Adapty matches subscribers by original transaction ID, and we don't count events from Apple webhook without exposing them to Adapty SDK (we technically can't do it).
2. **Used data will be there**. You'll have all Adapty profiles with user properties and can use them in Segments, and Profiles/CRM.

Follow our tutorial to send us historical data.

---

title: "Observer mode" description: ""

metadataTitle: ""

Adapty is a powerful and flexible in-app purchase platform designed to boost your revenue and subscriber base. With features like customizable paywalls tailored to specific user segments, A/B testing for pricing, duration, trial periods, and visual elements, as well as comprehensive analytical tools for app monetization and third-party integrations, Adapty empowers your growth strategy.

However, if you already have your own purchase infrastructure and aren't prepared to switch to Adapty's system, you can explore the Adapty Observer mode. This limited mode omits the use of Adapty paywalls, targeting them to user audiences, managing the subscriptions, including handling renewals and billing retries; and focusing solely on analytics. Despite its limitations, Observer mode still offers robust analytics capabilities, including integration with attribution systems, advanced analytics, messaging, and CRM profiles.

Both modes are offered at the same price and require your mobile app to be updated, so the choice essentially comes to either transitioning to Adapty's infrastructure for full functionality or retaining your current infrastructure while only gaining third-party integrations and analytical capabilities.

| Functionality | Observer mode | Full mode | |----------|----------|---------| **Comprehensive Analytics** | âœ… | âœ… | | **Third-Party Integrations** | âœ… | âœ… | | **Responding to purchase events to give/restrict paid access to your users** | âŒ | âœ… | | **Purchases Infrastructure Maintainer** | You | Adapty | | **A/B Testing** |

:warning:

Feasible, but requires a significant amount of additional coding and configuration, more than in Full Mode.

| âœ… | | **Implementation Time** |

For analytics and integrations: Less than an hour

With A/B tests: Up to a week with thorough testing

| Several hours |

How Observer mode works

In Observer mode, the Adapty SDK simply listens for new transactions from Apple/Google and forwards them to the Adapty backend. App developers are tasked with managing access to the paid content in their app, completing transactions, handling renewals, addressing billing issues, and so on.

How to set up Observer mode

1. Set up initial integration of Adapty with the Google Play and with the App Store.
2. Install and configure Adapty SDK. Make sure to set the `observerMode` parameter to `true`. Refer to our framework-specific instructions for iOS, for Android, for Flutter, for React Native, and for Unity.
3. (optional) If you want to use 3d-party integrations, set them up as described in the Configure 3d-party integration topic.

:::warning When operating in Observer mode, the Adapty SDK does not finalize transactions, so ensure you handle this aspect yourself. :::

How to use paywalls and A/B tests in Observer mode

In Observer mode, Adapty SDK cannot determine the source of purchases as you make them in your own infrastructure. Therefore, if you intend to use paywalls and/or A/B tests in Observer mode, you need to associate the transaction coming from your app store with the corresponding paywall in your mobile app code.

Additionally, paywalls designed with Paywall Builder should be displayed in a special way when using the Observer mode:

- Display paywalls in Observer mode for iOS or Android.
- Associate paywalls to purchase transactions in Observer mode.

---

title: "Migration from RevenueCat" description: "The document outlines a migration plan for switching from RevenueCat SDK to Adapty SDK, which involves learning the core differences, installing Adapty SDK, switching App Store server-side notifications, testing and releasing a new app version, and optionally importing historical data."

metadataTitle: ""

Your migration plan will have 5 logical steps and take an average of 2 hours. 90% of all migrations take less than one working day.

1. Learn the core differences; create and prepare an Adapty account *(5 minutes)*;
2. Install Adapty SDK for your platform (iOS, Android, React Native, Flutter, Unity) instead of RevenueCat SDK *(1 hour)*;
3. Set up Apple App Store server notifications to Adapty and (optionally) raw events forwarding *(5 minutes)*;
4. Test and release updates of your app *(30 minutes);*
5. (Optional) Ask RevenueCat support for historical data in CSV format *(5 minutes);*
6. (Optional) Import historical data via Adapty support *(30 minutes).*

:::info Your subscribers will migrate automatically

All users who have ever activated a subscription will instantly move to Adapty as soon as they open a new version of your app with Adapty SDK. The subscription status validation and premium access will be restored automatically. :::

Before you push a new version of your app with Adapty SDK, make sure to check our release Ñhecklist.

Learn the core differences; create and prepare an Adapty account

Adapty and RevenueCat SDKs are similarly designed. The biggest difference is the network usage and the speed: Adapty SDK is designed to provide you with information on demand as fast as possible when you ask for it. For example, when requesting a paywall, you get the remote config first to pre-build your onboarding or paywall and then request products in a dedicated request.

Naming is slightly different:

| RevenueCat | Adapty | | :---------- | :---------- | | Package | Product | | Offering | Paywall | | Paywall | Paywall Builder | | Entitlement | Access level |

Adapty has a concept of placement. It's a logical place inside your app where the user can make a purchase. In most cases, you have one or two placements:

- Onboarding (because 80% of all purchases take place there);
- General (you show it in settings or inside the app after the onboarding).

./ width: '300px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Install Adapty SDK and replace RevenueCat SDK

IInstall Adapty SDK for your platform (iOS, Android, React Native, Flutter, Unity) in your app.

You need to replace a couple of SDK methods on the app side. Let's look at the most common functions and how to replace them with Adapty SDK.

SDK activation

Replace `Purchases.configure` with `Adapty.activate`.

Getting paywalls (offerings)

Replace `Purchases.shared.getOfferings` with `Adapty.getPaywall`.

In Adapty, you always request the paywall via placement id. In practice, you only fetch no more than 1-2 paywalls, so we made this on purpose to speed up the SDK and reduce network usage.

Getting a user (customer profile)

Replace `Purchases.shared.getCustomerInfo` with `Adapty.getProfile`.

Getting products

In RevenueCat, you use the following structure:`Purchases.shared.getOfferings` and then `self.offering?.availablePackages`.

In Adapty, you first request a paywall (read above) to get immediate access to Adapty's remote config and then call for products with `Adapty.getPaywallProducts`.

Making a purchase

Replace `Purchases.shared.purchase` with `Adapty.makePurchase`.

Checking access level (entitlement)

Get a customer profile (read above first) and then replace

```
customerInfo?.entitlements["premium"]?.isActive == true
```

with

```
profile.accessLevels["premium"]?.isActive == true.
```

Restore purchase

Replace `Purchases.shared.restorePurchases` with `Adapty.restorePurchases`.

Check if the user is logged in

Replace `Purchases.shared.isAnonymous` with `if profile.customerUserId == nil`.

Log in user

Replace `Purchases.shared.logIn` with `Adapty.identify`.

Log out user

Replace `Purchases.shared.logOut` with `Adapty.logout`.

Switch App Store server-side notifications to Adapty

Read how to do this here.

Test and release a new version of your app

If you're reading this, you've already:

- [x] Configured Adapty Dashboard
- [x] Installed Adapty SDK
- [x] Replaced SDK logic with Adapty functions
- [x] Switched App Store server-side notifications to Adapty and optionally turn on raw events forwarding to RevenueCat
- [ ] Made a sandbox purchase
- [ ] Made a new app release

If you checked the points above, just make a test purchase in the Sandbox and then release the app.

:::info Go through release checklist

Make the final check using our list to validate the existing integration or add additional features such as attribution or analytics integrations. :::

(Optional) Ask RevenueCat support for historical data in CSV format

:::warning Don't rush the historical data import

You should wait for at least a week after the release with the SDK before doing historical data import. During that time we will get all the info about purchase prices from the SDK, so the data you import will be more relevant. :::

Ask RevenueCat about the historical data on their support page. For the file format reference, check this page: Importing Historical Data to Adapty. Also, you can use this Google Sheets file.

2022-03-16at_15.40.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Write us to import your historical data

Contact us using the website messenger or just email us at support@adapty.io with your CSV file.

FAQ

I successfully installed Adapty SDK and released a new app version with it. What will happen to my legacy subscribers who did not update to a version with Adapty SDK?

Most users charge their phones overnight, it's when the App Store usually auto-updates all their apps, so it shouldn't be a problem. There may still be a small number of paid subscribers who did not upgrade, but they will still have access to the premium content. You don't need to worry about it and force them to update.

Do I need to request historical data from RevenueCat as quickly as possible, or will I lose it?

You don't need to make it super fast; make a release with Adapty SDK first, and then give us your historical data. We will restore the history of your users' payments and fill in profiles and charts.

I use MMP (AppsFlyer, Adjust, etc.) and analytics (Mixpanel, Amplitude, etc.). How do I make sure that everything will work?

You first need to pass us the IDs of such 3rd party services via our SDK that you want us to send data to. Read the guide for attribution integration and for analytics integration. For historical data and legacy users, **make sure you pass us those IDs from the data export you asked for from RevenueCat.**

---

title: "Migration from Glassfy" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Glassfy services will be ending in December 2024. We worked with them to make the transition as easy as possible for you. This guide will help you migrate your subscribers to Adapty in less than a day. Most importantly, the migration will be 100% seamless for your customers; they will continue using the app without interruptions.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::info Moving from Glassfy? Get 6 months free of Pro+ plan

When you migrate from Glassfy to Adapty, you can use all our features, including Paywall Builder, A/B tests, ML predictions, and Targeting for free for the first 6 months â€” no strings attached. Just use this link to sign up. Try it for yourself and see why thousands of apps use Adapty to grow their revenue. :::

Here are the 3 easy steps to migrate your app from Glassfy to Adapty:

1. Learn the core differences (very few of them) and set up an Adapty account *(15 minutes)*;
2. Install Adapty SDK for your platform (iOS *(1 hour)*;
3. Test and release the new version of your app *(30 minutes)*.

:::info Your subscribers will migrate automatically

All users who have ever activated a subscription will instantly move to Adapty as soon as they open a new version of your app with the Adapty SDK. Subscription status validation and premium access will be restored automatically. :::

Learn the core differences and set up an Adapty account

Adapty and Glassfy SDKs are similarly designed. Adapty allows you to show different paywalls to different audiences, but it's optional.

Naming is slightly different:

| Glassfy | Adapty | | :---------- | :---------- | | SKU | Product | | Permission | Access level | | Offering | Paywall |

Creating an Adapty account

Create an account using a special link . You can also invite your colleagues.

Set up integration with the App Store and/or Google Play

You've done it at least once already, so we'll just leave the link to the docs. You will have to provide a Bundle ID and subscription keys and set up server notifications so that Adapty can work with purchases.

- Configuring subscription key and enabling Apple server notifications for the App Store
- Configuring service account key file and enabling Google server notifications for the Google Play

Create products

To sell the product in Adapty SDK, you have to create it in the dashboard first. This process is very similar to how SKUs are created in Glassfy. Just give it a name, choose the access level (aka permission), and product IDs for the App Store / Google Play. You can read more about the products here.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Create paywalls

Once you created the products, you should create the paywalls (aka offerings). A paywall can have one or more products. It can also have remote configuration, which allows you to customize the paywalls without new releases, localize the paywalls and onboarding and much more. You can even design and create paywalls without any coding with the Adapty paywall builder.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Create placements

Adapty has a concept of placement. It's a logical place inside your app where the user can make a purchase. In most cases, you have one or two placements:

- Onboarding (because 80% of all purchases take place there)
- General (you show it in settings or inside the app after the onboarding)

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

With the placements, you can dynamically change which Paywall or A/B test should be displayed in the designated place of your application. You can even show different paywalls to different audiences in your application.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Well done, now you can integrate Adapty SDK into your app!

Install Adapty SDK to replace Glassfy SDK

Install Adapty SDK for your platform (iOS.

SDK activation

Glassfy ```swift func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

Glassfy.initialize(apiKey: "YOUR*API*KEY", watcherMode: false)

[...]

// optionally login your user Glassfy.login(user: "youruser")

} </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin class App : Application() { override fun onCreate() { super.onCreate()

```
Glassfy.initialize(this, "YOUR_API_KEY", false, null)
```

} } </TabItem> <TabItem value="java" label="Java" default> java public class App extends Application { @Override public void onCreate() { super.onCreate(); Glassfy.initialize(this, "YOUR*API*KEY", false, null); } } </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try {

await Glassfy.initialize('YOU*API*KEY',watcherMode: false);

} catch (e) { // error [...] } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try {

await Glassfy.initialize('YOU*API*KEY', false);

} catch (e) { // initialization error } ```

Adapty

```swift func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(customerUserId: "YOUR*USER*ID") // optionally add your internal user id

```
Adapty.activate(with: configurationBuilder) { error in
// handle the error
}
```

} </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin override fun onCreate() { super.onCreate() Adapty.activate( applicationContext, AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withCustomerUserId(customerUserId) // optionally add your internal user id .build() ) } </TabItem> <TabItem value="java" label="Java" default> java @Override public void onCreate() { super.onCreate(); Adapty.activate( applicationContext, new AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withCustomerUserId(customerUserId) // optionally add your internal user id .build() ); } </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { Adapty().activate(); } on AdaptyError catch (adaptyError) {} } catch (e) {} </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript adapty.activate('PUBLIC*SDK*KEY', { customerUserId: 'YOUR*USER*ID', // optionally add your internal user id }); ```

**Fetch offerings (paywalls)**

**Glassfy**

```
swift Glassfy.offerings { (offerings, err) in if let offering = offerings?["premium"] { // display your offering's skus for sku in offering.skus { // sku.extravars // sku.product.localizedTitle // sku.product.localizedDescription // sku.product.price } } } kotlin Glassfy.offerings() { offers, err -> offers?.all ?.firstOrNull { it.offeringId == "premium" } ?.also { // display your offering's skus for (sku in it.skus) { // sku.extravars // sku.product.title // sku.product.description // sku.product.price } } } java Glassfy.offerings(new OfferingsCallback() { @Override public void onResult(@Nullable Offerings offers, @Nullable GlassfyError err) { Offering offering = null; if (offers != null) { for (Offering o : offers.getAll()) { if (o.getOfferingId().equals("premium")) { offering = o; } } } if (offering != null) { // display your offering's skus for (Sku sku : offering.getSkus()) { // sku.getExtravars(); // sku.getProduct().getTitle(); // sku.getProduct().getDescription(); // sku.getProduct().getPrice(); } } } }); ```javascript try { var offerings = await Glassfy.offerings(); var offering = offerings.all ?.singleWhere((offering) => offering.offeringId == 'premium');

offering?.skus?.forEach((sku) {
    // sku.product.description
    // sku.product.price
});

} catch (e) { // initialization error [...] } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { let offering = Glassfy.offerings().all.find((o) => o.offeringId === 'premium');

offering?.skus.forEach((sku) => {
  // sku.extravars
  // sku.product.description;
  // sku.product.price
});

} catch (e) { [...] } ```
```

Adapty

```
swift Adapty.getPaywall(placementId: "YOUR_PLACEMENT_ID", locale: "en") { result in switch result { case let .success(paywall): // the requested paywall // call getPaywallProducts here case let .failure(error): // handle the error } } kotlin Adapty.getPaywall("YOUR_PLACEMENT_ID", locale = "en") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall // call getPaywallProducts here } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getPaywall("YOUR*PLACEMENT*ID", "en", result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue(); // the requested paywall // call getPaywallProducts here

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final paywall = await Adapty().getPaywall(id: "YOUR*PLACEMENT*ID", locale: "en"); // the requested paywall // call getPaywallProducts here } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const id = 'YOUR*PLACEMENT*ID'; const locale = 'en';

const paywall = await adapty.getPaywall(id, locale);

// the requested paywall // call getPaywallProducts here } catch (error) { // handle the error } ```

swift Adapty.getPaywallProducts(paywall: paywall) { result in switch result { case let .success(products): // the requested products array for product in products { // product.localizedTitle // product.localizedDescription // product.localizedPrice // product.localizedSubscriptionPeriod // product.price } case let .failure(error): // handle the error } } kotlin Adapty.getPaywallProducts(paywall) { result -> when (result) { is AdaptyResult.Success -> { val products = result.value // the requested products for (product in products) { // product.localizedTitle // product.localizedDescription // product.localizedPrice // product.localizedSubscriptionPeriod // product.price } } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getPaywallProducts(paywall, result -> { if (result instanceof AdaptyResult.Success) { List products = ((AdaptyResult.Success>) result).getValue(); // the requested products for (AdaptyPaywallProduct product: products) { // product.localizedTitle // product.localizedDescription // product.localizedPrice // product.localizedSubscriptionPeriod // product.price }

```
} else if (result instanceof AdaptyResult.Error) {
```

```
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}
```

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final products = await Adapty().getPaywallProducts(paywall: paywall); // the requested products array for (var product in products!) { // product.localizedTitle // product.localizedDescription // product.price.amount // product.price.localizedString // product.subscriptionDetails.localizedSubscriptionPeriod } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { // ...paywall const products = await adapty.getPaywallProducts(paywall); // the requested products list } catch (error) { // handle the error } ```

In Adapty, you always request the paywall via placement id. If you want to see conversions, learn how to log paywall views.

Check permissions (access level)

Glassfy

```
swift Glassfy.permissions { permissions, err in guard let permissions = permissions else { return } for p in permissions.all { switch (p.permissionId) { case "aPermission": if (p.isValid) { // unlock aFeature } break; default: print("Permission not handled"); break; } } } kotlin Glassfy.permissions { permission, err -> // update app status accordingly permission?.all?.forEach { when (it.permissionId) { "premium" -> if (it.isValid) { // unlock aFeature } else -> println("Permission not handled"); } } } java Glassfy.permissions(new PermissionsCallback() { @Override public void onResult(@Nullable Permissions permission, @Nullable GlassfyError error) { // update app status accordingly if (permission != null) { for (Permission p: permission.getAll()) { switch (p.getPermissionId()) { case "premium": if (p.isValid()) { // unlock aFeature } break; default: Log.d(TAG, "Permission not handled"); } } } } }); javascript try { var permission = await Glassfy.permissions(); permission.all?.forEach((p)=> { if (p.permissionId == "premium" && p.isValid==true) { // unlock aFeature } }); } catch (e) { // initialization error [...] } ```typescript try { const permissions = await Glassfy.permissions(); permissions.all.forEach((p)=>{ switch (p.permissionId) { case "premium": if (permission.isValid) { // unlock } break;

        default:
            break;
    }
});
```

} catch (e) { // initialization error [...] } ```

```
swift Adapty.getProfile { result in if let profile = try? result.get() { // check the access profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive ?? false { // grant access to premium features } } } kotlin Adapty.getProfile { result -> when (result) { is AdaptyResult.Success -> { val profile = result.value // check the access } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getProfile(result -> { if (result instanceof AdaptyResult.Success) { AdaptyProfile profile = ((AdaptyResult.Success) result).getValue(); // check the access

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final profile = await Adapty().getProfile(); // check the access } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const profile = await adapty.getProfile(); } catch (error) { // handle the error } ``` **Adapty**

Make a purchase

Glassfy

```
swift Glassfy.purchase(sku: premiumSku) { (transaction, e) in // update app status accordingly if let p = transaction?.permissions["aPermission"] { if p.isValid { // unlock aFeature } else { // lock aFeature } } } kotlin Glassfy.purchase(activity, sku) { transaction, err -> // update app status accordingly transaction?.permissions ?.all ?.firstOrNull { it.permissionId == "aPermission" } ?.also { if (it.isValid) { // unlock aFeature } else { // lock aFeature } } } java Glassfy.purchase(activity, sku, new PurchaseCallback() { @Override public void onResult(@Nullable Transaction t, @Nullable GlassfyError err) { // update app status accordingly Permission permission = null; if (t != null) { for (Permission p : t.getPermissions().getAll()) { if (p.getPermissionId().equals("aPermission")) { permission = p; } } } if (permission != null) { if (permission.isValid()) { // unlock aFeature } else { // lock aFeature } } } }); ```javascript try { var transaction = await Glassfy.purchaseSku(sku);

var p = transaction.permissions?.all?.singleWhere((permission) => permission.permissionId == 'premium');
if (p?.isValid==true) {
    // unlock aFeature
}
else {
    // lock aFeature
}
```

} catch (e) { // initialization error [...] } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const transaction = await Glassfy.purchaseSku(premiumSku); const permission = transaction.permissions.all.find((p) => p.permissionId === "aPermission"); if (permission && permission.isValid) { // unlock aFeature } } catch (e) { // initialization error [...] } ```

Adapty

```
swift Adapty.makePurchase(product: product) { result in switch result { case let .success(info): if info.profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive ?? false { // successful purchase } case let .failure(error): // handle the error } } ```kotlin Adapty.makePurchase(activity, product) { result -> when (result) { is AdaptyResult.Success -> { val info = result.value //NOTE: info is null in case of cross-grade with DEFERRED proration mode val profile = info?.profile

        if (profile?.accessLevels?.get("YOUR_ACCESS_LEVEL")?.isActive == true) {
            // grant access to premium features
        }
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
    }
}
```

} </TabItem> <TabItem value="java" label="Java" default> java Adapty.makePurchase(activity, product, result -> { if (result instanceof AdaptyResult.Success) { AdaptyPurchasedInfo info = ((AdaptyResult.Success) result).getValue(); //NOTE: info is null in case of cross-grade with DEFERRED proration mode AdaptyProfile profile = info != null ? info.getProfile() : null;

```
    if (profile != null) {
        AdaptyProfile.AccessLevel premium = profile.getAccessLevels().get("YOUR_ACCESS_LEVEL");

        if (premium != null && premium.isActive()) {
            // successful purchase
        }
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); ```

```
javascript try { final profile = await Adapty().makePurchase(product: product); if (profile?.accessLevels['YOUR_ACCESS_LEVEL']?.isActive ?? false) { // successful purchase } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } ```typescript try { const profile = await adapty.makePurchase(product); const isSubscribed = profile?.accessLevels['YOUR*ACCESS*LEVEL']?.isActive;

if (isSubscribed) {
    // grant access to features in accordance with access level
}
```

} catch (error) { // handle the error } ```

Test and release a new version of your app

If you're reading this, you've already:

- [x] Configured integration with the App Store / Google Play
- [x] Configured Adapty Dashboard
- [x] Installed Adapty SDK
- [x] Replaced Glassfy with Adapty functions
- [ ] Made a sandbox purchase
- [ ] Made a new app release

If you checked the points above, just make a test purchase in the Sandbox and then release the app.

:::info Go through release checklist

Make the final check using our list to validate the existing integration or add additional features such as attribution or analytics integrations. :::

FAQ

I successfully installed Adapty SDK and released a new app version with it. What will happen to my legacy subscribers who did not update to a version with Adapty SDK?

Most users charge their phones overnight, it's when the App Store usually auto-updates all their apps, so it shouldn't be a problem. There may still be a small number of paid subscribers who did not upgrade, but they will still have access to the premium content. You don't need to worry about it and force them to update.

Do I need to request historical data from Glassfy as quickly as possible, or will I lose it?

You don't need to make it super fast; make a release with Adapty SDK first, and then give us your historical data. We will restore the history of your users' payments and fill in profiles and charts.

I use MMP (AppsFlyer, Adjust, etc.) and analytics (Mixpanel, Amplitude, etc.). How do I make sure that everything will work?

You first need to pass us the IDs of such 3rd party services via our SDK that you want us to send data to. Read the guide for attribution integration and for analytics integration.

---

title: "Importing historical data to Adapty" description: ""

metadataTitle: ""

After installing the Adapty SDK and releasing your app, you can access your users and subscribers in the Profiles section. But what if you have legacy infrastructure and need to migrate to Adapty, or simply want to see your existing data in Adapty?

:::note Data import is not mandatory

Adapty will automatically grant access levels to historical users and restore their purchase events once they open the app with the Adapty SDK integrated. For this use case, importing historical data is not necessary. However, importing data ensures precise analytics if you have a significant number of historical transactions, although it is generally not required for migration. :::

To import data to Adapty:

1. Export your transactions to a CSV file (separate files should be provided for iOS, Android, and Stripe). Please refer to the Import file format section below for detailed requirements.
2. If any file exceeds 1 GB, prepare a data sample with approximately 100 lines.
3. Upload all the files to Google Drive (you can compress them, but keep them separate).
4. For iOS transactions, ensure the **In-app purchase API** section in the **App settings** is filled out with the **Issuer ID**, **Key ID**, and the **Private key** (.P8 file) even if you use the StoreKit 1. See the Provide Issuer ID and Key ID and Upload In-App Purchase Key file sections for detailed instructions.
5. Share the links with our team via email or through the online chat in the Adapty Dashboard.

Do not worry, importing historical data will not create duplicates, even if that data overlaps with existing entries in Adapty.

Known limitations for Android

1. Only active subscriptions will be restored; expired transactions will not be.
2. Only the latest renewals in a subscription will be restored; the entire chain of purchases will not be.
3. If the product price has changed since the purchase, the current price will be used, which may result in incorrect pricing.

Import file format

Please prepare your data in a file or files that meet the following rules:

- [ ] The file format is .CSV.
- [ ] Separate files for Android, iOS, and Stripe imports.
- [ ] Every import file contains all required columns.
- [ ] The columns in the import file(s) have headers.
- [ ] The column headers are exactly as in the **Column name** column in the table below. Please check for typos.
- [ ] Columns that are not required can be absent from the file. Don't add empty columns for data you don't have.
- [ ] Import files should not have extra columns not mentioned in the table. If present, please delete them.
- [ ] Values are separated by commas.
- [ ] Values are not enclosed in quotes.
- [ ] If there are several **apple*original*transaction*id*'s** for one user, add all of them as separate lines for each **appleoriginal*transaction*id**. Otherwise, we may not be able to restore consumable purchases.

Please use the following files as samples for iOS and Android.

Available import file columns

| Column name | Presence | Description | |----------|--------|----------| | **user*id*** | required | ID of your user | | **appleoriginal*transaction*id** | required for iOS |

The original transaction ID or OTID (learn more), used in StoreKit 2 import mechanism. As one user can have multiple OTIDs, it is enough to provide at least one for successful import.

Note: We require In-app purchase API credentials for this import to be set up in your Adapty Dashboard. Learn how to do it here .

| | **google*product*id** | required for Google | Product ID in the Google Play Store. | | **google*purchase*token** | required for Google | A unique identifier that represents the user and the product ID for the in-app product they purchased | | **google*is*subscription** | required for Google | Possible values are `1` \| `0` | | **stripe*token*** | required for Stripe | Token of a Stripe object that represents a unique purchase. Could either be a token of Stripe's Subscription (`sub_...`) or Payment Intent (`pi_...`). | | **subscriptionexpiration*date*** | optional | The date of subscription expiration, i.e. next charging date, date, and time with timezone (2020-12-31T23:59:59-06:00) | | **createdat** | optional | Date and time of profile creation (2019-12-31 23:59:59-06:00) | | **birthday** | optional | The birthday of the user in format 2000-12-31 | | **email** | optional | The e-mail of your user | | **gender** | optional | The gender of the user. Possible values are: `f` \| `m` | | **phone*number*** | optional | The phone number of your user | | **country** | optional | format ISO 3166-1 alpha-2 | | **firstname** | optional | The first name of your user | | **last*name*** | optional | The last name of your user | | **lastseen** | optional | The date and time with timezone (2020-12-31T23:59:59-06:00) | | **idfa** | optional | The identifier for advertisers (IDFA) is a random device identifier assigned by Apple to a user's device. Applicable to iOS apps only | | **idfv** | optional | The identifier for vendors (IDFV) is a unique code assigned to all apps developed by a single developer, which in this case refers to your apps. Applicable to iOS apps only | | **advertising*id*** | optional | The Advertising ID is a unique code assigned by the Android Operating System that advertisers might use to uniquely identify a user's device | | **amplitudeuser*id*** | optional | The user ID from Amplitude | | **amplitudedevice*id*** | optional | The device ID from Amplitude | | **mixpaneluser*id*** | optional | User ID from Mixpanel | | **appmetricaprofile*id*** | optional | User profile ID from AppMetrica | | **appmetricadevice*id*** | optional | The device ID from AppMetrica | | **appsflyerid** | optional | Unique identifier from AppsFlyer | | **adjust*device*id** | optional | The device ID from Adjust | | **facebook*anonymous*id** | optional | A unique identifier generated by Facebook for users who interact with your app or website anonymously, meaning they are not logged into Facebook | | **branch*id*** | optional | Unique identifier from Branch | | **attributionsource** | optional | The source integration of the attribution, for example, appsflyer | adjust | branch | apple*search*ads | | **attributions*tatus*** | optional | organic | nonorganic|unknown | | **attribution*channel*** | optional | The attribution channel that brought the transaction | | **attributioncampaign** | optional | The attribution campaign that brought the transaction | | **attribution*ad*group** | optional | The attribution ad group that brought the transaction | | **attribution*ad*set** | optional | The attribution ad set that brought the transaction | | **attribution_creative** | optional | Specific visual or textual elements used in an advertisement or marketing campaign that are tracked to determine their effectiveness in driving desired actions, such as clicks, conversions, or installs |

Required Fields

There are 2 groups of required fields for each platform: **user_id** and data identifying purchases specific to the corresponding platform. Refer to the table below for the mandatory fields per platform.

| Platform | Required fields | |--------|---------------| | iOS |

user*id*

appleoriginal*transaction*id

| | Android |

user*id*

googleproduct*id*

googlepurchase*token*

googleis*subscription*

| | Stripe |

userid

stripe_token

|

Without these fields, Adapty won't be able to fetch transactions.

For precise cohort analytics, please specify `created_at`. If not provided, we will assume the install date to be the same as the first purchase date.

Import data to Adapty

Please contact us and share your import files via support@adapty.io or through the online chat in the Adapty Dashboard.

---

title: "Initial integration with the App Store" description: "Get started with Adapty's initial integration process with the App Store, ensuring seamless connectivity and enabling access to Adapty's powerful features for optimizing user engagement. Explore step-by-step instructions for integrating your mobile app with Adapty and start leveraging its capabilities today"

metadataTitle: "Adapty Initial Integration with App Store: A Quick Guide"

We're thrilled to have you on board with Adapty! Our priority is to help you hit the ground running and achieve the best possible outcomes for your app. This guide is designed to get you started with Adapty if your app is available in the App Store.

Integrating Adapty into your mobile app involves establishing connections between your app and Adapty at both the App Store and SDK levels. Though it may seem hard on the surface, following the onboarding in Adapty Dashboard or these instructions will help you accomplish this in no more than 30 minutes.

Guide for the initial integration

- [ ] Once you create an account in Adapty and provide your mobile app name and category, we set up the app for you within our Adapty platform.
- [ ] Generate In-App Purchase Key in the App Store Connect
- [ ] Configure App Store integration itself in the Adapty dashboard and App Store Connect
- [ ] Enable App Store server notifications in the App Store Connect
- [ ] Install AdaptySDKs for the frameworks you're using:
  - [ ] Install Adapty SDKs for native iOS
  - [ ] Install Adapty SDKs for Flutter
  - [ ] Install Adapty SDKs for React Native
  - [ ] Install Adapty SDKs for Unity
- [ ] Build your application and run it in sandbox mode.

After the initial integration is complete, you can begin using Adapty's features.

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to make changes to your app's code. Specifically, you need todisplay the paywalls at least and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist . This will ensure that you've completed all the necessary steps before your app goes live with Adapty SDK onboard. :::

---

title: "Generate In-App Purchase Key in App Store Connect" description: "Optimize in-app purchase transactions and enhance security by generating an In-App Purchase Key in App Store Connect, facilitating secure communication with Apple's servers. Learn how to streamline the validation process for app purchases and align with Apple's focus on improving security measures"

metadataTitle: "App Store Connect: Generating In-App Purchase Key"

The **In-App Purchase Key** is a specialized API key created within App Store Connect to validate the purchases by confirming their authenticity.

:::note To generate API keys for the App Store Server API, you you must hold either an Admin role or an Account Holder role in App Store Connect. You can also read about how to generate API Keys in the Apple Developer Documentation. :::

1. Open **App Store Connect**. Proceed to **Users and Access â†' Integrations â†' In-App Purchase** section.
2. Then click the add button **(+)** next to the **Active** title.

in-appkey.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened **Generate In-App Purchase Key** window, enter the name of the key for your future reference. It will not be used in Adapty.
4. Click the **Generate** button. Once the **Generate in-App Purchase Key** window closes, you'll see the created key in the **Active** list.

inappfile.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

5. Once you've generated your API key, click the **Download In-App Purchase Key** button to obtain the key as a file.

in-apppurchase_key.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. In the **Download in-App Purchase Key** window, click the **Download** button. The file is saved to your computer.
   It's crucial to keep this file secure for future uploading to the Adapty Dashboard. Note that the generated file can only be downloaded once, so ensure safe storage until you upload it. The generated .p8 key from the **In-App Purchase section** can be used for both the In-app purchase API and promotional offers.

---

title: "Configure App Store integration" description: "Streamline App Store integration for your mobile app with Adapty, ensuring seamless validation of purchases and subscription updates. Learn how to input your app's configuration data from the App Store during initial onboarding or make changes later in the App Settings of the Adapty Dashboard"

metadataTitle: "Adapty App Store Integration Configuration"

This section describes how to establish the connection between the App Store and Adapty for your iOS app. This is required for us to be able to show subscription analytics and validate purchases. You can complete the integration during the initial onboarding or later in the **App Settings** within the Adapty Dashboard.

Although you might have initially configured the integration of your mobile app and Adapty during onboarding, you can modify these settings later in the **App settings**.

:::danger Configuration changes can be done safely during the Sandbox phase, until your mobile app goes live with Adapty SDK installed. Changes after the release can break the purchase flow in your app. :::

Step 1. Provide Bundle ID

Bundle ID is the unique identifier of your app in the App Store. This is required for the basic functionality of Adapty, such as subscription processing.

1. Open App Store Connect. Select your app and proceed to **General** â†' **App Information** section.
2. Copy the **Bundle ID** in the **General Information** sub-section.

idapple.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Open the **App settings -> iOS SDK** tab from the Adapty top menu.

settingstop_menu.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Paste the copied value to the **Bundle ID** field.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Step 2. Provide Issuer ID and Key ID

The **In-app purchase Issuer ID**, referred to as **Issuer ID** in App Store Connect, is a special ID that identifies the issuer who created the authentication token. The **In-App Purchase Key ID**, referred to as **Key ID** in App Store Connect, is a unique identifier associated with a cryptographic key you've generated in the Generate In-App Purchase Key in App Store Connect section.

1. Open **App Store Connect**. Proceed to **Users and Access â†' Integrations â†' In-App Purchase** section.
2. In the *Active *list, find the key you've created in the Generate In-App Purchase Key in App Store Connect section.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Copy **Issuer ID** and paste it to the **In-app purchase Issuer ID** field in the Adapty Dashboard.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Copy the **Key ID** and paste it to the **In-app purchase Key ID** and **Subscription key ID** fields in the Adapty Dashboard.

Step 3. Upload In-App Purchase Key file

Upload the **In-App Purchase Key** file you've downloaded in the Generate In-App Purchase Key in App Store Connect section

into the **Private key (.p8 file)** and **Subscription (.p8 file)** fields in the Adapty Dashboard.

Step 4. Enter App Store shared secret

The **App Store shared secret**, also known as the App Store Connect Shared Secret, is a 32-character hexadecimal string used for in-app purchases and subscription receipt validation.

1. Open App Store Connect. Select your app and proceed to **General** â†' **App Information** section.
2. Scroll down to the **App-Specific Shared Secret** sub-section.

:::info If the **App-Specific Shared Secret** sub-section is absent, make sure you have an Account Holder or Admin role. If you have an Admin role and yet cannot see the **App-Specific Shared Secret** sub-section, ask the Account Holder of the app (the person who has created the application in the App Store Connect) to generate the App Store shared secret for the app. After that, the sub-section will be shown to Admins as well. :::

3. Click the **Manage** button.

4. In the opened **App-Specific Shared Secret** window, copy the **Shared Secret**. If no shared secret is visible, first click either the **Manage** or **Generate** button whichever is available, and then copy the **Shared Secret**.
5. Paste the copied **Shared Secret** to the **App Store shared secret** field in the Adapty Dashboard.

6. Click the **Save** button in the Adapty Dashboard to confirm the changes.

---

title: "Enable App Store server notifications" description: "Learn how to set up App Store server notifications in Adapty to receive real-time updates on refunds and other events directly from the App Store"

metadataTitle: "How to Enable App Store Server Notifications in Adapty"

Setting up App Store server notifications is crucial for ensuring data accuracy as it enables you to receive updates instantly from the App Store, including information on refunds and other events.

1. Copy the **URL for App Store server notification** in the Adapty Dashboard.

2. Open App Store Connect. Select your app and proceed to **General** â†' **App Information** section, **App Store Server Notifications** subsection.
3. Paste the copied **URL for App Store server notification** into the **Production Server URL** and **Sandbox Server URL** fields.

---

title: "Initial integration with Google Play" description: "Get started with Adapty's initial integration process with the Googple Play Store, ensuring seamless connectivity and enabling access to Adapty's powerful features for optimizing user engagement. Explore step-by-step instructions for integrating your mobile app with Adapty and start leveraging its capabilities today"

metadataTitle: "Adapty Initial Integration with Google Play Store: A Quick Guide"

We're thrilled to have you on board with Adapty! Our priority is to help you hit the ground running and achieve the best possible outcomes. This guide is designed to get you started with Adapty if your app is available in the Google Play Store.

Integrating Adapty into your mobile app involves establishing connections between your app and Adapty at both Google Play and SDK levels. While the process may appear extensive, following the built-in onboarding in the Adapty Dashboard or the instructions below will simplify it, typically taking no more than 1 hour.

Checklist for the initial integration

- [ ] Once you create an account in Adapty and provide your mobile app name and category, we set up the app for you within our Adapty platform.
- [ ] Enable Developer APIs in Google Cloud Console
- [ ] Create a service account in the Google Cloud Console
- [ ] Grant permissions to the service account in the Google Play Console
- [ ] Generate the service account key file in the Google Cloud Console
- [ ] Configure Google Play integration itself in the Adapty Dashboard
- [ ] Enable Real-time developer notifications (RTDN) in the Google Play Console
- [ ] Install and configure AdaptySDKs (you may install SDKs for one or more frameworks, whatever are needed)
  - [ ] Install Adapty SDKs for Android
  - [ ] Install Adapty SDKs for Flutter
  - [ ] Install Adapty SDKs for React Native
  - [ ] Install Adapty SDKs for Unity
- [ ] Build your application and run it. Running as a snapshot or in a sandbox environment is sufficient.

:::note It takes at least 24 hours for changes to take effect but there's a hack. In Google Play Console, open any application and in the **Monetize** section go to **Products** -> **Subscriptions**/**In-app products**. Change the description of any product and save the changes. Everything should be working now, you can revert in-app changes. :::

After the initial integration is complete, you can begin using Adapty's features.

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to make changes to your app's code. Specifically, you need todisplay the paywalls at least and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

---

title: "Enable Developer APIs in Google Play Console" description: "Optimize your subscription model by enabling Developer APIs in the Google Play Console for seamless integration with Adapty. Learn how to activate Developer APIs to facilitate automated processes and real-time data analysis for enhanced subscription management"

metadataTitle: "Google Play Console: Enabling Developer APIs for Adapty Integration"

If your mobile app is available in the Play Store, activating Developer APIs is crucial for integrating it with Adapty. This step ensures seamless communication between your app and our platform, facilitating automated processes and real-time data analysis to optimize your subscription model. The following APIs should be enabled:

- Google Play Android Developer API
- Google Play Developer Reporting API
- Cloud Pub/Sub API

If your app isn't distributed via the Play Store, you can skip this step. However, if you do sell through the Play Store, you can delay this step for now, though it's crucial for Adapty's basic functionality. After completing the onboarding process, you can configure the application store settings in the **App settings** section.

Here's how to enable Developer APIs in the Google Play Console:

1. Open the Google Cloud Console.

2. In the top-left corner of the Google Cloud window, select the project you wish to use or create a new one. Ensure you use the same Google Cloud project until you upload the service account key file to Adapty.

cloudproject.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Open the **Google Play Android Developer API** page.

playapi.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Click the **Enable** button and wait for the status **Enabled** to show. This means the Google Android Developer API is enabled.

playapi*create*credentials.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

5. Open the **Google Play Developer Reporting API** page.

playdeveloper*reporting*api.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. Click the **Enable** button and wait for the status **Enabled** to show.

playdeveloper*reporting*api_enabled.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

7. Open the **Cloud Pub/Sub API** page.

CloudPub*Sub*API.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

8. Click the **Enable *button and wait for the status* *Enabled** to show.

PubSub*API*enabled.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Developer APIs are enabled.

You can recheck it in the **APIs & Services** page of teh Google Cloud Console. Scroll the page down and validate the table at the bottom of the page contains all 3 APIs:

- Google Play Android Developer API
- Google Play Developer Reporting API
- Cloud Pub/Sub API

enabledapi.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Create service account in the Google Cloud Console" description: "Enhance integration with Adapty by creating a service account in the Google Cloud Console, facilitating streamlined data access automation and seamless connectivity with the Google Play Console. Learn how to create a service account to optimize your app management process"

metadataTitle: "Google Cloud Console: Creating a Service Account for Adapty"

For Adapty to automate data access, a service account is necessary in the Google Play Console.

1. Open **IAM & Admin - > Service accounts** section of the Google Cloud Console. Make sure you use the correct project.

cloudcreate*service*account.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the Click the **Create service account** button.

accountdetails.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In **Service account details** sub-section of the **Create service account** window, enter the **Service Account Name** you want. We recommend including "Adapty" in the name to indicate the purpose of this account. The **Service account ID** will be created automatically.

4. Copy the service account email address and save it for future usage.

5. Click the **Create and continue** button.

accessto_project.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. In the **Select a role** drop-down list of the **Grant this service account access to project** sub-section, select **Pub/Sub -> Pub/Sub Admin**. This role is required to enable real-time developer notifications.

accountrole.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

7. Click the **Add another role** button.

8. In a new **Role** drop-down list, select **Monitoring -> Monitoring Viewer**. This role is required to allow monitoring of the notification queue.

9. Click the **Continue** button.

useraccess.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

10. Click the **Done** button without any changes. The **Service accounts** window opens.

---

title: "Grant permissions to service account in the Google Play Console" description: "Authorize Adapty's service account by granting essential permissions in the Google Play Console, facilitating seamless management of subscriptions and validation of purchases. Learn how to grant permissions to optimize your app's integration with Adapty"

metadataTitle: "Google Play Console: Granting Permissions to Adapty's Service Account"

Grant the required permissions to the service account that Adapty will use to manage subscriptions and validate purchases.

1. Open the **Users and permissions** page in the Google Play Console and click the **Invite new users** button.

andpermissions.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the **Invite user** page, enter the email of the service users you've created.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Switch to the **Account permissions** tab.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Select the following permissions:
    - View app information and download bulk reports (read-only)
    - View financial data, orders, and cancellation survey responses
    - Manage orders and subscriptions
    - Manage store presence
5. Click the **Invite user** button.
6. In the **Send invite?** window, click the **Send invite** button. The service account will show in the user list.

---

title: "Generate service account key file in the Google Play Console" description: "Enhance app security and establish a secure link between your Play Store mobile application and Adapty by generating service account key files in the Google Play Console. Learn how to generate key files to ensure the security of your app and prevent unauthorized access"

metadataTitle: "Google Play Console: Generating Service Account Key Files for Adapty"

To link your mobile app on the Play Store with Adapty, you'll need to generate special service account key files in the Google Play Console and upload them to Adapty. These files help secure your app and prevent unauthorized access.

:::warning It usually takes at least 24 hours for your new service account to become active. However, there's a hack. After creating the service account in the Google Play Console, open any application and navigate to **Monetize** -> **Products** -> **Subscriptions/In-app products**. Edit the description of any product and save the changes. This should activate the service account immediately, and you can revert the changes afterward. :::

1. Open the **Service accounts** section in the Google Play Console. Ensure youâ€™ve selected the correct project.

managekeys.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the window that opens, click **Add key** and choose **Create new key** from the dropdown menu.

newkey.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the **Create private key for [Your*project*name]** window, click **Create**. Your private key will be saved to your computer as a JSON file. You can find it using the file name provided in the **Private key saved to your computer** window.

privatekey.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. In the **Create private key for Your*project*name** window, click the **Create** ***button. This action will save your private key on your computer as a JSON file. You can use the name of the file provided in the opened*** **Private key saved to your computer** window to locate it if needed.

keysaved.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Youâ€™ll need this file when you initially integrate Adapty with Google Play.

:::warning It usually takes at least 24 hours for your new service account to become active. However, there's a hack. After creating the service account in the Google Play Console, open any application and navigate to **Monetize** -> **Products** -> **Subscriptions/In-app products**. Edit the description of any product and save the changes. This should activate the service account immediately, and you can revert the changes afterward.

. :::

---

title: "Configure Google Play Store integration" description: "Set your app's integration with Adapty by configuring Play Store integration with Adapty. Learn how to input your app's configuration data for seamless validation of purchases and receipt of subscription updates within Adapty's platform"

metadataTitle: "Google Play Store Integration Configuration with Adapty"

This section outlines the integration process for your mobile app sold via Google Play with Adapty. You'll need to input your app's configuration data from the Play Store into the Adapty Dashboard. This step is crucial for validating purchases and receiving subscription updates from the Play Store within Adapty.

You can complete this process during the initial onboarding or make changes later in the **App Settings** of the Adapty Dashboard.

:::danger Configuration change is only acceptable until you release your mobile app with integrated Adapty paywalls. The change after the release will break the integration and the paywalls will stop showing in your mobile app. :::

Step 1. Provide Package name

The Package name is the unique identifier of your app in the Google Play Store. This is required for the basic functionality of Adapty, such as subscription processing.

1. Open the Google Play Developer Console.
2. Select the app whose ID you need. The **Dashboard** window opens.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Find the product ID under the application name and copy it.

4. Open the **App settings** from the Adapty top menu.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

5. In the **Android SDK** tab of the **App settings** window, paste the copied **Package name**.

Step 2. Upload the account key file

1. Upload the service account private key file in JSON format that you have created at the Create service account key file step into the **Service account key file** area.

keyfile.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Don't forget to click the **Save** button to confirm the changes.

---

title: "Enable Real-time developer notifications (RTDN) in Google Play Console" description: "Stay informed about critical events and maintain data accuracy by enabling Real-time Developer Notifications (RTDN) in the Google Play Console for Adapty. Learn how to set up RTDN to receive instant updates about refunds and other important events from the Play Store"

metadataTitle: "Google Play Console: Enabling Real-time Developer Notifications (RTDN) for Adapty"

Setting up real-time developer notifications (RTDN) is crucial for ensuring data accuracy as it enables you to receive updates instantly from the Play Store, including information on refunds and other events.

1. Open the **App settings** from the Adapty top menu.

settingstop_menu.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Copy the contents of the **Enable Pub/Sub API** field next to the **Google Play RTDN topic name** title.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note If the contents of the **Enable Pub/Sub API** field have a wrong format (correct format starts with `projects/...`), make sure you've enabled all developers APIs and granted all required permissions to the service account. :::

3. Open the Google Play Console, choose your app, and scroll down the left menu to find **Monetize** -> **Monetization setup**.
4. In the **Google Play Billing** section, select the **Enable real-time notifications** check-box.

topicname.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

5. Paste the contents of the **Enable Pub/Sub API** field you've copied in the Adapty **App Settings** into the **Topic name** field.
6. Click the **Save changes** button in the Google Play Console.

---

title: "Installation of Adapty SDKs" description: "Learn how to integrate AdaptySDK and AdaptyUI SDK into your mobile app for seamless functionality and effortless creation of subscription purchase pages with the Paywall builder. Get detailed installation and configuration guidance tailored for various frameworks"

metadataTitle: "AdaptySDK and AdaptyUI SDK Integration Guide for Mobile Apps"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Main **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

The Adapty SDK installation and configuration depend on your framework, so refer to the following doc topics for detailed guidance:

- Install and configure Adapty SDKs on iOS
- Install and configure Adapty SDKs on Android
- Install and configure Adapty SDKs on Flutter
- Install and configure Adapty SDKs on React Native
- Install and configure Adapty SDKs on Unity

---

title: "iOS - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on iOS, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "iOS - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI SDK version | | :---------- | :---------- | | 2.7.x, 2.8.x | 2.0.x | | 2.9.x - 2.10.0 | 2.1.2 | | 2.10.1 | 2.1.3 | | 2.10.3 and all later 2.10.x versions | 2.1.5 | | 2.11.1 | 2.11.1 | | 2.11.2 | 2.11.2 | | 2.11.3 | 2.11.3 | | 3.x | Included as a module within the Adapty SDK. For more information, see iOS - Adapty SDK v. 3.x installation & configuration. |

You can install AdaptySDK and AdaptyUI SDK via CocoaPods, or Swift Package Manager.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install SDKs via CocoaPods

1. Add Adapty to your `Podfile`:

   ```
   shell title="Podfile" pod 'Adapty', '~> 2.11.3' pod 'AdaptyUI', '~> 2.11.3'
   ```

2. Run:

   ```
   sh title="Shell" pod install
   ```

This creates a `.xcworkspace` file for your app. Use this file for all future development of your application.

Install SDKs via Swift Package Manager

1. In Xcode go to **File** -> **Add Package Dependency...**. Please note the way to add package dependencies can differ in XCode versions. Refer to XCode documentation if necessary.
2. Enter the repository URL `https://github.com/adaptyteam/AdaptySDK-iOS.git`
3. Choose the version, and click the **Add package** button. Xcode will add the package dependency to your project, and you can import it.
4. In the **Choose Package Products** window, click the **Add package** button once again. The package will appear in the **Packages** list.
5. Repeat steps 2-3 for AdaptyUI SDK URL: `https://github.com/adaptyteam/AdaptyUI-iOS.git`.

Configure Adapty SDK

You only need to configure the Adapty SDK once, typically early in your application lifecycle:

```swift // In your AppDelegate class:

let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) // optional .with(customerUserId: "YOUR*USER*ID") // optional .with(idfaCollectionDisabled: false) // optional .with(ipAddressCollectionDisabled: false) // optional

Adapty.activate(with: configurationBuilder) { error in // handle the error } </TabItem> <TabItem value="SwiftUI" label="SwiftUI" default> swift import Adapty

@main struct SampleApp: App { init() let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) // optional .with(customerUserId: "YOUR*USER*ID") // optional .with(idfaCollectionDisabled: false) // optional .with(ipAddressCollectionDisabled: false) // optional

    Adapty.activate(with: configurationBuilder) { error in
      // handle the error
    }
}

var body: some Scene {
    WindowGroup {
        ContentView()
    }
}

} ```
```

Parameters:

| Parameter | Presence | Description | |---------|--------|----------| **PUBLIC*SDK*KEY** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection | | **observerMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics.

The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional | An identifier of the user in your system. We send it in subscription and analytical events, to attribute events to the right profile. You can also find customers by `customerUserId` in the **Profiles and Segments** menu. | | **idfaCollectionDisabled** | optional |

Set to `true` to disable IDFA collection and sharing.

the user IP address sharing.

The default value is `false`.

For more details on IDFA collection, refer to the Analytics integration section.

| | **ipAddressCollectionDisabled** | optional |

Set to `true` to disable user IP address collection and sharing.

The default value is `false`.

|

:::note - Note, that StoreKit 2 is available since iOS 15.0. Adapty will implement the legacy logic for older versions. - Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only. - **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one. :::

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` at any time in the application's lifespan, but we recommend that you do this before configuring Adapty.

```
swift title="Swift" Adapty.logLevel = .verbose
```

Redirect the logging system messages

If you for some reason need to send messages from Adapty to your system or save them to a file, you can override the default behavior:

```
swift title="Swift" Adapty.setLogHandler { level, message in writeToLocalFile("Adapty \(level): \(message)") }
```

title: "Android - Adapty SDK Installation and configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Android, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided"

metadataTitle: "Android - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI version | | :----------------- | :--------------- | | 2.7.xâ€"2.9.x | 2.0.x | | 2.10.0 | 2.1.2 | | 2.10.2 | 2.1.3 | | 2.11.x | 2.11.x |

You can install Adapty SDK via Gradle.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install via Gradle

`groovy dependencies { ... implementation 'io.adapty:android-sdk:2.11.3' implementation 'io.adapty:android-ui:2.11.2' } kotlin dependencies { ... implementation("io.adapty:android-sdk:2.11.3") implementation("io.adapty:android-ui:2.11.1") }` ```toml //libs.versions.toml

[versions] .. adapty = "2.11.3" adaptyUi = "2.11.1"

[libraries] .. adapty = { group = "io.adapty", name = "android-sdk", version.ref = "adapty" } adapty-ui = { group = "io.adapty", name = "android-ui", version.ref = "adaptyUi" }

//module-level build.gradle.kts

dependencies { ... implementation(libs.adapty) implementation(libs.adapty.ui) } ```

If the dependency is not being resolved, please make sure that you have `mavenCentral()` in your Gradle scripts.

▶ The instruction on how to add it

If your project doesn't have `dependencyResolutionManagement` in your `settings.gradle`, add the following to your top-level `build.gradle` at the end of repositories:

`groovy title="top-level build.gradle" allprojects { repositories { ... mavenCentral() } }`

Otherwise, add the following to your `settings.gradle` in `repositories` of `dependencyResolutionManagement` section:

`groovy title="settings.gradle" dependencyResolutionManagement { ... repositories { ... mavenCentral() } }`

Configure Proguard

You should add `-keep class com.adapty.** { *; }` to your Proguard configuration.

Configure Adapty SDK

Add the following to your `Application` class:

```kotlin override fun onCreate() { super.onCreate() Adapty.activate( applicationContext, AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withObserverMode(false) //default false .withCustomerUserId(customerUserId) //default null .withIpAddressCollectionDisabled(false) //default false .build() )

```
//OR
//the method is deprecated since Adapty SDK v2.10.5

Adapty.activate(applicationContext, "PUBLIC_SDK_KEY", observerMode = false, customerUserId = "YOUR_USER_ID")
```

} ```

```java @Override public void onCreate() { super.onCreate(); Adapty.activate( applicationContext, new AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withObserverMode(false) //default false .withCustomerUserId(customerUserId) //default null .withIpAddressCollectionDisabled(false) //default false .build() );

```
//OR
//the method is deprecated since Adapty SDK v2.10.5

Adapty.activate(getApplicationContext(), "PUBLIC_SDK_KEY", false, "YOUR_USER_ID");
```

} ```

Configurational options:

| Parameter | Presence | Description | |---------|--------|-----------| **PUBLIC*SDK*KEY** | required |

The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings**-> **General** tab -> **API keys** subsection.

Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only.

| | **observerMode** | optional |

A boolean value that controls Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional | An identifier of the user in your system. We send it in subscription and analytical events, to attribute events to the right profile. You can also find customers by `customerUserId` in the **Profiles and Segments** menu. If you don't have a user ID at the time of Adapty initialization, you can set it later using `.identify()` method. Read more in the Identifying users section. | | **IpAddressCollectionDisabled** | optional |

A boolean parameter. Set to `true` to disable the collection of the user IP address. The default value is `false`.

Parameter works with `AdaptyConfig.Builder` only.

|

:::note **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one. :::

Set up the logging system

Adapty logs errors and other important information to help you understand what is going on. There are the following levels available:

| Level | Description | | :---------------------- | :-- | | `AdaptyLogLevel.NONE` | Nothing will be logged. Default value | | `AdaptyLogLevel.ERROR` | Only errors will be logged | | `AdaptyLogLevel.WARN` | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | `AdaptyLogLevel.INFO` | Errors, warnings, and various information messages will be logged. | | `AdaptyLogLevel.VERBOSE` | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set the log level in your app before configuring Adapty.

`kotlin Adapty.logLevel = AdaptyLogLevel.VERBOSE java Adapty.setLogLevel(AdaptyLogLevel.VERBOSE);`

Redirect the logging system messages

If you for some reason need to send messages from Adapty to your system or save them to a file, you can override the default behavior:

```
kotlin Adapty.setLogHandler { level, message -> //handle the log } java Adapty.setLogHandler((level, message) -> { //handle the log });
```

---

title: "Flutter - Adapty SDK Installation and configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Flutter, enabling seamless integration of Adapty into your mobile app . Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "Flutter - Adapty SDK Installation and Configuration Guide"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI SDK version | | :----------------- | :------------------- | | 2.9.3 | 2.1.0 | | 2.10.0 | 2.1.1 | | 2.10.1 | 2.1.2 | | 2.10.2 | 2.1.3 |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install Adapty SDKs

1. Add Adapty and AdaptyUI to your `pubspec.yaml` file:

   ```
   yaml title="pubspec.yaml" dependencies: adapty_flutter: ^2.10.1 adapty_ui_flutter: ^2.1.1
   ```

2. Run:

   ```
   bash title="Bash" flutter pub get
   ```

3. Import Adapty SDKs in your application in the following way:

   ```
   dart title="Dart" import 'package:adapty_flutter/adapty_flutter.dart'; import 'package:adapty_ui_flutter/adapty_ui_flutter.dart';
   ```

Configure Adapty SDKs

The configuration of the Adapty SDK for Flutter slightly differs depending on the mobile operating system (iOS or Android) you are going to release it for.

Configure Adapty SDKs for iOS

Create `Adapty-Info.plist` and add it to your project. Add the flag `AdaptyPublicSdkKey` in this file with the value of your Public SDK key.

```
xml title="Adapty-Info.plist" <dict> <key>AdaptyPublicSdkKey</key> <string>PUBLIC_SDK_KEY</string> <key>AdaptyObserverMode</key> <false/> </dict>
```

Parameters:

| Parameter | Presence | Description | |---------|--------|-----------| |**AdaptyPublicSdkKey** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection | | **AdaptyObserverMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. At any purchase or restore in your application, you'll need to call `.restorePurchases()` method to record the action in Adapty. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **idfaCollectionDisabled** | optional |

A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

Configure Adapty SDKs for Android

1. Add the `AdaptyPublicSdkKey` flag into the appâ€™s `AndroidManifest.xml` (Android) file with the value of your Public SDK key.

   ```
   xml title="AndroidManifest.xml" <application ...> ... <meta-data android:name="AdaptyPublicSdkKey" android:value="PUBLIC_SDK_KEY" /> <meta-data
   android:name="AdaptyObserverMode" android:value="false" /> </application>
   ```

   Required parameters:

| Parameter | Presence | Description | |---------|--------|-----------| | PUBLIC*SDK*KEY | required |

Contents of the **Public SDK key** field in the **App Settings -> General** tab in the Adapty Dashboard. **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one.

Make sure you use the **Public SDK key** for Adapty initialization, since the **Secret key** should be used for server-side API only.

| | AdaptyObserverMode | optional |

A boolean value that is controlling Observer mode . Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics.

The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | AdaptyIDFACollectionDisabled | optional |

A boolean parameter, that allows you to disable IDFA collection for your app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

2. In your application, add:

   ```
   javascript title="Flutter" import 'package:adapty_flutter/adapty_flutter.dart';
   ```

3. Activate Adapty SDK with the following code:

   ```
   javascript title="Flutter" try { Adapty().activate(); } on AdaptyError catch (adaptyError) {} } catch (e) {}
   ```

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` in your app before configuring Adapty.

```
javascript title="Flutter" try { await Adapty().setLogLevel(AdaptyLogLevel.verbose); } on AdaptyError catch (adaptyError) { } catch (e) {}
```

:::danger Read checklist before releasing the app

Before releasing your application, go through the Release Checklist to ensure that you have completed all the steps, and also check the success of the integration using the criteria for assessing its success. :::

---

title: "React Native - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on React Native, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "React Native - - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

You currently need to have a `react-native-adapty` of version 2.4.7 or higher to use UI SDK.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI version | | :----------------- | :--------------- | | 2.7.0 â€“ 2.9.2 | 2.0.0 - 2.0.1 | | 2.9.3 - 2.9.8 | 2.1.0 | | 2.10.0 | 2.1.1 | | 2.10.1 | 2.1.2 | | 2.11.x | 2.11.0 |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install Adapty SDKs

Currently, React Native provides two development paths: Expo and Pure React Native. Adapty seamlessly integrates with both. Please refer to the section below that matches your chosen setup.

Install Adapty SDKs for Expo React Native

You can streamline your development process with Expo Application Services (EAS). While configuration may vary based on your setup, here you'll find the most common and straightforward setup available:

1. If you haven't installed the EAS Command-Line Interface (CLI) yet, you can do so by using the following command:

   ```
   sh title="Shell" npm install -g eas-cli
   ```

2. In the root of your project, install the dev client to make a development build:

   ```
   sh title="Shell" expo install expo-dev-client
   ```

3. Run the installation command:

   ```
   sh title="Shell" expo install react-native-adapty expo install @adapty/react-native-ui
   ```

4. For iOS: Make an iOS build with EAS CLI. This command may prompt you for additional info. You can refer to expo official documentation for more details:

   ```
   sh title="Shell" eas build --profile development --platform ios
   ```

5. For Android: Make an Android build with EAS CLI. This command may prompt you for additional info. You can refer to expo official documentation for more details:

   ```
   sh title="Shell" eas build --profile development --platform android
   ```

6. Start a development server with the following command:

   ```
   sh title="Shell" expo start --dev-client
   ```

This should result in the working app with react-native-adapty.

Possible errors:

| Error | Description | |-----|----------| |`Failed to start (Invariant Violation: Native module cannot be null)` |

if you scan a QR code from a CLI dev client it might lead you to this error. To resolve it you can try the following:

> On your device open EAS built app (it should provide some Expo screen) and manually insert the URL that Expo provides (screenshot below). You can unescape special characters in URL with the JS function `unescape(â€œstringâ€)`, which should result in something like `http://192.168.1.35:8081`

|

Install Adapty SDKs with Pure React Native

If you opt for a purely native approach, please consult the following instructions:

1. In your project, run the installation command:

   ```
   sh title="Shell" yarn add react-native-adapty yarn add @adapty/react-native-ui
   ```

2. For iOS: Install required pods:

   ```
   sh title="Shell" pod install --project-directory=ios pod install --project-directory=ios/
   ```

   The minimal supported iOS version is 13.0. If you encounter an error during pod installation, locate this line in your`ios/Podfile` and update the minimal target. Tand update the minimum target. Afterward, you should be able to successfully execute `pod install`.

   ```
   diff title="Podfile" -platform :ios, min_ios_version_supported +platform :ios, 13.0
   ```

3. For Android: Update the `/android/build.gradle` file. Make sure there is the `kotlin-gradle-plugin:1.8.0` dependency or a newer one:

   ```
   groovy title="/android/build.gradle" ... buildscript { ... dependencies { ... classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.8.0" } } ...
   ```

Configure Adapty SDKs

To use Adapty SDKs, import `adapty` and invoke `activate` in your *core component* such as `App.tsx`. Preferably, position the activation before the React component to ensure no other Adapty calls occur before the activation.

```typescript title="/src/App.tsx" import { adapty } from 'react-native-adapty';

adapty.activate('PUBLIC*SDK*KEY');

const App = () => { // ... } ```

You can pass several optional parameters during activation:

`typescript adapty.activate('PUBLIC_SDK_KEY', { observerMode: false, customerUserId: 'YOUR_USER_ID', logLevel: 'error', __debugDeferActivation: false, ipAddressCollectionDisabled: false, ios: { idfaCollectionDisabled: false, }, }); ```javascript import { IosStorekit2Usage, LogLevel } from 'react-native-adapty';

adapty.activate('PUBLIC*SDK*KEY', { observerMode: false, customerUserId: 'YOUR*USER*ID', logLevel: LogLevel.ERROR, __debugDeferActivation: false, ipAddressCollectionDisabled: false, ios: { idfaCollectionDisabled: false, }, }); ```

Activation parameters:

| Parameter | Presence | Description | |---------|--------|----------| **PUBLIC*SDK*KEY** | required |

A Public SDK Key is the unique identifier used to integrate Adapty into your mobile app. You can copy it in the Adapty Dashboard: **App settings -> General *tab -> *API Keys** section.
```

SDK keys are unique for every app, so if you have multiple apps make sure you choose the right one.

Make sure you use the **Public SDK key** for the Adapty initialization, since the **Secret key** should be used for the server-side API only.

| | **observerMode** | optional |

A boolean value controlling Observer mode . Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional |

An identifier of a user in your system. We send it with subscription and analytical events, so we can match events to the right user profile. You can also find customers using the `customerUserId` in the Profiles section.

If you don't have a user ID when you start with Adapty, you can add it later using the `adapty.identify()` method. For more details, see the Identifying users section.

| | **logLevel** | optional | A string parameter that makes Adapty record errors and other important information to help you understand what's happening. | | **__debugDeferActivation** | optional | A boolean parameter, that lets you delay SDK activation until your next Adapty call. This is intended solely for development purposes and **should not be used in production**. | | **ipAddressCollectionDisabled** | optional |

Set to `true` to disable user IP address collection and sharing.

The default value is `false`.

| | **idfaCollectionDisabled** | optional | A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`. For more details, refer to the Analytics integration section. |

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` at any time in the application's lifespan, but we recommend that you do this before configuring Adapty.

`typescript adapty.setLogLevel('verbose');` ```javascript import { LogLevel } from 'react-native-adapty';

adapty.setLogLevel(LogLevel.VERBOSE); ```

For both `activate` and `setLogLevel` methods TypeScript validates the string you pass as an argument. However, if you're using JavaScript, you may prefer to use the `LogLevel` enum, that would guarantee to provide you a safe value:

Handle logs

If you're storing your standard output logs, you might wish to distinguish Adapty logs from others. You can achieve this by appending a prefix to all `AdaptyError` instances that are logged:

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.prefix = "[ADAPTY]"; ```

You can also handle all raised errors from any location you prefer using `onError`. Errors will be thrown where expected, but they will also be duplicated to your event listener.

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.onError = error => { // ... console.error(error); }; ```

## Delay SDK activation for development purposes

Adapty pre-fetches all necessary user data upon SDK activation, enabling faster access to fresh data.

However, this may pose a problem in the iOS simulator, which frequently prompts for authentication during development. Although Adapty cannot control the StoreKit authentication flow, it can defer the requests made by the SDK to obtain fresh user data.

By enabling the `__debugDeferActivation` property, the activate call is held until you make the next Adapty SDK call. This prevents unnecessary prompts for authentication data if not needed.

It's important to note that **this feature is intended for development use only**, as it does not cover all potential user scenarios. In production, activation should not be delayed, as real devices typically remember authentication data and do not repeatedly prompt for credentials.

Here's the recommended approach for usage:

`typescript title="Typescript" adapty.activate('PUBLIC_SDK_KEY', { __debugDeferActivation: isSimulator(), // 'isSimulator' from any 3rd party library });`

---

title: "Unity - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Unity, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

## metadataTitle: "Unity - - Adapty SDK Installation and Configuration Guide"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| AdaptySDK-Unity version | AdaptyUI-Unity version | | :---------------------- | :---------------------- | | 2.7.1 | 2.0.1 | | 2.9.0 | not compatible |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

## Install Adapty SDKs

To install the Adapty SDKs:

1. Download the `adapty-unity-plugin-*.unitypackage` from GitHub and import it into your project.

./ width: '400px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Download the `adapty-ui-unity-plugin-*.unitypackage` from GitHub and import it into your project.

./ width: '400px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Download and import the External Dependency Manager plugin.
4. The SDK uses the "External Dependency Manager" plugin to handle iOS Cocoapods dependencies and Android gradle dependencies. After the installation, you may need to invoke the dependency manager:

   `Assets -> External Dependency Manager -> Android Resolver -> Force Resolve`

   and

   `Assets -> External Dependency Manager -> iOS Resolver -> Install Cocoapods`

5. When building your Unity project for iOS, you would get `Unity-iPhone.xcworkspace` file, which you have to open instead of `Unity-iPhone.xcodeproj`, otherwise, Cocoapods dependencies won't be used.

## Configure Adapty SDKs

To configure the Adapty SDKs for Unity, start by initializing the Adapty Unity Plugin and then using it as described in the guidance below. Additionally, ensure to set up your logging system to receive errors and other important information from Adapty.

### Initiate Adapty Unity Plugin on iOS

The Adapty Unity Plugin on iOS is initialized automatically. To make it work properly:

1. Manually create the `Adapty-Info.plist` file and add it to the `/Assets` folder of your Unity project. It will be automatically copied to the Xcode project during the build phase. Below is an example of how this file should be structured:

```
xml title="Xml" <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>AdaptyPublicSdkKey</key> <string>insert_here_your_Adapty_public_key#</string> <key>AdaptyObserverMode</key> <false/> </dict> </plist>
```

Parameters:

| Parameter | Presence | Description | |---------|--------|-----------| |**AdaptyPublicSdkKey** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection | | **AdaptyObserverMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **idfaCollectionDisabled** | optional |

A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

Initiate Adapty Unity Plugin on Android

The Adapty Unity Plugin on Android is automatically initialized. To ensure it works properly:

1. Add `<meta-data` section with "AdaptyPublicSdkKey" as a direct child of the `<application` section to your project's AndroidManifest.xml file. If you don't have one, it can be easily created in **Project Settings** -> **Player** -> **Settings for Android** -> **Publishing settings** -> **Custom Main Manifest** checkbox). Here is an example:

```xml title="Xml"

    <meta-data
        android:name="AdaptyPublicSdkKey"
        android:value="PUBLIC_SDK_KEY" />

     <meta-data
         android:name="AdaptyObserverMode"
         android:value="false" />
</application>

```

Configurational options:

| Parameter | Presence | Description | |---------|--------|-----------| |**AdaptyPublicSDKkey** | required |

The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection.

Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only.

| | **AdaptyObservermode** | optional |

A boolean value that controls Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. Default value is `false`.

Note: When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

|

Use Adapty Unity Plugin

1. Create a script to listen to Adapty events. Name it `AdaptyListener` in your scene. We suggest using the `DontDestroyOnLoad` method for this object to ensure it persists throughout the application's lifespan.

adaptylistener.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty uses `AdaptySDK` namespace. At the top of your script files that use the Adapty SDK, you may add

```
csharp title="C#" using AdaptySDK;
```

2. Subscribe to Adapty events:

```csharp title="C#" using UnityEngine; using AdaptySDK;

public class AdaptyListener : MonoBehaviour, AdaptyEventListener { void Start() { DontDestroyOnLoad(this.gameObject); Adapty.SetEventListener(this); }

 public void OnLoadLatestProfile(Adapty.Profile profile) {
   // handle updated profile data
 }

} ```
```

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other important information to help you understand what is going on. There are three levels available:

| error | Only errors will be logged. | | :------ | :---------- | | warn | Errors and messages from the SDK that do not cause critical errors are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can call `SetLogLevel()` method in your app before configuring Adapty.

---

title: "Products" description: "Learn about App Store and Google Play products â€“ digital items users can purchase within apps, ranging from in-game currency to premium features. Discover how Adapty revolutionizes mobile app monetization and user engagement with its seamless product management capabilities. With Adapty, consolidate similar products across platforms into a single internal product, streamlining management and maximizing revenue potential. Elevate user experience, increase app revenue, and attract more users effortlessly with Adapty's innovative solutions"

metadataTitle: "What are products in mobile apps and how to improve mobile app monetization with Adapty"

A Product is any item or content that is available for purchase within a mobile application. This can include various digital goods such as virtual items, subscriptions, additional features, or content upgrades that users can buy to enhance their experience within the app. For example, in a gaming app, products might include in-game currency, power-ups, or expansion packs. In a productivity app, products might include premium features or access to exclusive content. These products are managed and sold through the in-app purchase system provided by the platform (e.g., Apple App Store or Google Play Store).

In Adapty you can combine similar products that you have in App Store and Play Store in a single internal product. This allows you to use a single Adapty product across all platforms, instead of using each vendor's products.

Make sure you've done the initial configuration without skipping a single step and created the products in the App Store and/or created products in Google Play before adding them in the Adapty Dashboard.

:::note Checklist to successfully display products in your mobile app

1. Create products in the Adapty Dashboard.
2. Create a paywall in the Adapty Dashboard and add products to it
3. Show paywalls using the placements they belong to in your mobile app. :::

After you create products in the Adapty Dashboard, they are visible in the **Products** tab of the **Paywalls and Products** section.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Create product" description: "Learn how to optimize mobile app monetization with Adapty's product creation guide. Explore the seamless integration of App Store and Google Play products into a unified offering within Adapty. Simplify management and unlock revenue potential with Adapty's innovative approach to product creation. Get started today"

metadataTitle: "What is product in Adapty and how to create it"

No matter how you use Adapty, you need to create a product in the Adapty Dashboard and link products you've created in the app stores into it. Product creation in app stores is done separately from Adapty and involves defining details like price, duration, and free trials for your in-app purchases or subscriptions. Adapty will then use these settings to manage and analyze transactions in your app. Please check our guides on how to create products in stores:

- How to create a product in App Store
- How to create a product in Google Play

After your products are set up in the stores, you are ready to add your products to the Adapty Dashboard. To add a new product to your app:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **Create product** button located in the top-right corner of the product list page. This action will initiate the process of creating a product within your app. Adapty supports all types of products: subscriptions, non-consumable (including lifetime), and consumable.

2023-07-28at_16.38.192x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened **Create product** window, enter the following data:

   - **Product name**: enter the name of the product to be used in the Adapty dashboard. The name is primarily for your reference, so feel free to choose a name that is most convenient for you to use across the Adapty Dashboard.
   - **Access Level**: Select the access level to which the product belongs. The access level is used to determine the features unlocked after purchasing the product. Note that this list contains only previously created access levels. The `premium` access level is created in Adapty by default, but you can also add more access levels.
   - **Period**: select the duration of the subscription from the list. It should match the period configured in the App Store or Play Store. > ðŸš§ By default, all the products have an **Uncategorized** period. Make sure to set the correct one, otherwise, there might be problems with granting access to your customers.
     > If the product is not a subscription, use the following options:
     - **Lifetime**: Use a lifetime period for the products that unlock the premium features of the app forever.
     - **Non-Subscriptions**: For the products that are not subscriptions and therefore have no duration, use non-subscriptions. These can be unlocked for additional features, consumable products, etc.
     - **Consumables**: Consumable items can be purchased multiple times. They could be used up during the life of the application. Examples are in-game currency and extras. Please consider that consumable products donâ€™t affect access levels.

4. Next, let's configure the product information from each store:

   1. **App Store:**

      - **App Store Product ID:** This unique identifier is used to access your product on devices. To obtain the product ID for the App Store, please follow the steps outlined in the Product in App Store page, where you'll find detailed instructions on how to create and retrieve the product ID.

   2. **Play Store:**

      - **Play Store Product ID:** These are identifiers for the product from the Play Store. You need to provide at least one of them, but you can always add another one later if needed. To obtain the product ID for the Play Store, please follow the steps outlined in the Product in Play Store page, where you'll find detailed instructions on how to create and retrieve the product ID.
      - **Base Plan ID:** This ID is used to define the base plan for the product in the Play Store. When adding a subscription's Product ID on the Play Store you have to provide a Base Plan ID. A base plan defines the essential details of a subscription, encompassing the billing period, renewal type (auto-renewing or prepaid), and the associated price. Please note, that within Adapty, each combination of the same subscription and different base plans is treated as a separate product.
      - **Legacy fallback product**: A fallback product is used exclusively for apps using older versions of the Adapty SDK (versions 2.5 and below). By marking a product as backward compatible in the Google Play Console, Adapty can identify whether it can be purchased by older SDK versions. For this field please specify the value in the following format `<subscription_id>:<base_plan_id>`.

2023-07-28at_16.40.362x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

By configuring the product information for both the App Store and Play Store, you'll ensure smooth integration and effective management of your in-app purchases or subscriptions using Adapty.

5. (optional) You can create offers for the product if you need to.

6. Click the **Save** button to confirm the creation of the product.

---

title: "Edit product" description: ""

metadataTitle: "Edit product in Adapty"

In Adapty you can combine similar products that you have in App Store and Play Store in a single internal Product. This allows you to use a single Adapty product across all platforms, instead of using each vendor's products.

:::warning While you have the option to edit any product, it's crucial to ensure that making changes to products already used in live paywalls doesn't lead to discrepancies in your analytics.

Editing period, access level, App Store Product ID, and Play Store Product ID is not recommended because it may affect analytics clarity. Only edit them if you made a mistake, like setting the wrong period or typo in the product ID.

If you no longer use the product and want to replace it with another one, we strongly advise you to create a new product and update Paywalls and A/B tests accordingly. :::

To edit the product:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab.
2. Click the **3-dot** button next to the product and select the **Edit** option.
3. In the opened **Edit** window, make the changes you need. For more details on the options in this window, please read the Create product section.
4. Click the **Save** button to confirm the changes.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Delete product" description: ""

metadataTitle: ""

You can only delete products that are not used in paywalls.

To delete the product:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab.
2. Click the **3-dot** button next to the product and select the **Delete** option.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the **Delete product** window, enter the product name you're about to delete.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Click the **Delete forever** button to confirm the deletion.

---

title: "Add product to paywall" description: ""

metadataTitle: ""

To make a product visible and selectable within a paywall for your app's users, follow these steps:

1. While configuring a paywall, click the **Add product** button under the **Products** title.
2. From the opened drop-down list, select the products that will be shown to your customers. The list contains only previously created products. The order of the products is preserved on the SDK side, so it's important to consider the desired order when configuring the paywall. Additionally, you can specify an offer for a product if desired.

3. Click the **Save as draft** or **Save and publish** button depending on the status of the paywall.

Please keep in mind that after creation, it is not recommended to edit, add, or delete products to the paywall as this may affect the paywall metrics.

---

title: "Offers" description: "Unlock the potential of your mobile app's monetization strategy with Adapty's guide on adding offers to paywalls. Increase customer volume and retention by seamlessly integrating enticing offers into your app's paywalls. Learn how targeted promotions and incentives can elevate engagement and drive growth. Dive into our comprehensive guide now to maximize your app's revenue potential"

metadataTitle: "What are offers and how to use them in Adapty to increase your customer volume"

Offers in the App Store and Google Play are special deals or discounts provided by these platforms for in-app purchases.

There are 2 types of offers:

- **Introductory offer**
  An introductory offer is a special welcome for users who are exploring a subscription-based app for the first time. It's a promotion that you can set up to provide new subscribers with a discounted price, a free trial, or other enticing deals for a certain period.
  Please also consider, that introductory offers on iOS are applied automatically if the user is eligible, no need to create them in Adapty.
- **Promotional offer**
  A promotional offer is a friendly invitation for users who are already familiar with a subscription-based app. It's a special deal or discount that you can create to engage existing or past subscribers. With promotional offers, you can provide discounted prices, free trials, or other enticing deals to encourage users to renew or re-subscribe.

These offers help attract and keep users engaged, making the app experience more rewarding. By using these special incentives, you can boost user interest and loyalty, contributing to the overall success of their apps.

:::note Checklist for Adapty to successfuly process offers from the App Store and Play Store:

1. Create offers in the App Store Connect or create offers in the Google Play Console
2. (for iOS apps only) Upload a special In-App Purchase Key from App Store Connect to Adapty.
3. Create offers in Adapty
4. Add these offers to a paywall in Adapty :::

---

title: "Offers" description: "Unlock the potential of your mobile app's monetization strategy with Adapty's guide on adding offers to paywalls. Increase customer volume and retention by seamlessly integrating enticing offers into your app's paywalls. Learn how targeted promotions and incentives can elevate engagement and drive growth. Dive into our comprehensive guide now to maximize your app's revenue potential"

metadataTitle: "What are offers and how to use them in Adapty to increase your customer volume"

Offers in the App Store and Google Play are special deals or discounts provided by these platforms for in-app purchases.

There are 2 types of offers:

- **Introductory offer**
  An introductory offer is a special welcome for users who are exploring a subscription-based app for the first time. It's a promotion that you can set up to provide new subscribers with a discounted price, a free trial, or other enticing deals for a certain period.
  Please also consider, that introductory offers on iOS are applied automatically if the user is eligible, no need to create them in Adapty.
- **Promotional offer**
  A promotional offer is a friendly invitation for users who are already familiar with a subscription-based app. It's a special deal or discount that you can create to engage existing or past subscribers. With promotional offers, you can provide discounted prices, free trials, or other enticing deals to encourage users to renew or re-subscribe.

These offers help attract and keep users engaged, making the app experience more rewarding. By using these special incentives, you can boost user interest and loyalty, contributing to the overall success of their apps.

:::note Checklist for Adapty to successfuly process offers from the App Store and Play Store:

1. Create offers in the App Store Connect or create offers in the Google Play Console
2. (for iOS apps only) Upload a special In-App Purchase Key from App Store Connect to Adapty.
3. Create offers in Adapty
4. Add these offers to a paywall in Adapty :::

---

title: "Create offer" description: "Learn how to use Google Play and App Store offers to attract and keep users engaged in Adapty"

metadataTitle: "How to create offers in Adapty"

Adapty allows you to offer discounted pricing to existing or churned subscribers. To use this feature, you need to first create the offer in App Store Connect and/or create the offer in Play Console. Once you have the offer ready in the app stores, you can easily add it to Adapty:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab.

2. Find the product to which you want to add an offer and in the **Actions** column, click the **3-dot** button next to the product and select the **Edit** option.
3. In the opened **Edit product** window, click the **Add Offer** button below the **Offers** title.

4. In the added new empty line, enter the offer you wish to add to the product.

   Here are the fields for the offer:

   - **Offer name:** Provide a name for the offer to help identify it within Adapty. Use whatever name is convenient for you.
   - **App Store Offer ID:** This is the unique identifier of the offer you set in the App Store.
   - **Play Store Offer ID:** Similarly, this is the unique identifier of the offer you set in the Play Store.
5. Click the **Save** button to save the newly added offers to the product.

---

title: "Add offer to paywall" description: "Learn how to make an offer visible and selectable within a Adapty paywall for your app's users"

metadataTitle: "How to use App Store and Google Play offers to empower your paywalls"

Offers in the App Store and Google Play are special deals or discounts provided by these platforms for in-app purchases.

To make an offer visible and selectable within a paywall for your app's users, follow these steps:

1. While configuring the products on a paywall, click the **Add offer** button next to the product the offer belongs to. The button is available only for the products that have offers.

2. Select an offer you created earlier for this product from the **Offer** list.

---

title: "Paywalls" description: "Discover the power of Adapty's dynamic in-app product display solution. Learn how paywalls serve as versatile storefronts within your app, allowing you to tailor product offerings based on user subscriptions, location, and device type. With Adapty, enhance customer satisfaction and boost revenue by offering targeted product sets to different user segments effortlessly."

metadataTitle: "Unlock Revenue Growth: Adapty's Dynamic In-App Product Display Solution"

A paywall serves as an in-app storefront where customers can browse and make purchases. What sets it apart is its dynamic nature, allowing you to modify it without requiring app updates or even introducing new paywalls to users based on various factors using Placements.

There are two ways to design the view of a paywall: using the Paywall Builder, which is an easy and flexible no-code tool designed to help you create top-performing paywalls without requiring development or design skills; and the Remote Config, a powerful tool that allows you to build a paywall using JSON. In both scenarios, you'll have impressive paywalls for your users.

The way you show these paywalls in your mobile app depends on the tool you decide to pick:

1. Paywalls designed with the Paywall Builder include both what to display and how to display it. They can also process most user actions such as making purchases, opening links, or clicking buttons with no development required. Refer to the Design paywalls with Paywall Builder for details on designing paywalls with the Paywall Builder and Display paywalls created by Paywall Builder topic for details on presenting them in your mobile app.
2. On the other hand, paywalls customized using remote config can be tailored to your specific requirements, granting you complete freedom to design and process them as needed. Refer to the Design paywall with remote config topic for details on designing paywalls with the remote config and to the Display paywalls created by remote config topic for details on presenting them in your mobile app.

Paywall states

Paywalls can have four distinct states:

- **Draft**: These paywalls are in the preparation stage has never been used in any placements or A/B tests. Use this state while you are designing your paywall until you are ready to use it in your mobile app.
- **Live**: These paywalls are currently active and running in placements and/ or A/B tests. Live paywalls can be used in multiple A/B tests and associated with various placements. You may conduct one or more A/B tests based on a live paywall if it turns out to be effective.
- **Inactive**: These paywalls were previously active in placements but are currently not live. You can repurpose an inactive paywall for a new A/B test or choose to archive it if it is no longer required.
- **Archived**: These paywalls are no longer in use and have been archived. You can always restore the archived paywall to an active state.

---

title: "Paywalls" description: "Discover the power of Adapty's dynamic in-app product display solution. Learn how paywalls serve as versatile storefronts within your app, allowing you to tailor product offerings based on user subscriptions, location, and device type. With Adapty, enhance customer satisfaction and boost revenue by offering targeted product sets to different user segments effortlessly."

metadataTitle: "Unlock Revenue Growth: Adapty's Dynamic In-App Product Display Solution"

A paywall serves as an in-app storefront where customers can browse and make purchases. What sets it apart is its dynamic nature, allowing you to modify it without requiring app updates or even introducing new paywalls to users based on various factors using Placements.

There are two ways to design the view of a paywall: using the Paywall Builder, which is an easy and flexible no-code tool designed to help you create top-performing paywalls without requiring development or design skills; and the Remote Config, a powerful tool that allows you to build a paywall using JSON. In both scenarios, you'll have impressive paywalls for your users.

The way you show these paywalls in your mobile app depends on the tool you decide to pick:

1. Paywalls designed with the Paywall Builder include both what to display and how to display it. They can also process most user actions such as making purchases, opening links, or clicking buttons with no development required. Refer to the Design paywalls with Paywall Builder for details on designing paywalls with the Paywall Builder and Display paywalls created by Paywall Builder topic for details on presenting them in your mobile app.
2. On the other hand, paywalls customized using remote config can be tailored to your specific requirements, granting you complete freedom to design and process them as needed. Refer to the Design paywall with remote config topic for details on designing paywalls with the remote config and to the Display paywalls created by remote config topic for details on presenting them in your mobile app.

Paywall states

Paywalls can have four distinct states:

- **Draft**: These paywalls are in the preparation stage has never been used in any placements or A/B tests. Use this state while you are designing your paywall until you are ready to use it in your mobile app.
- **Live**: These paywalls are currently active and running in placements and/ or A/B tests. Live paywalls can be used in multiple A/B tests and associated with various placements. You may conduct one or more A/B tests based on a live paywall if it turns out to be effective.
- **Inactive**: These paywalls were previously active in placements but are currently not live. You can repurpose an inactive paywall for a new A/B test or choose to archive it if it is no longer required.
- **Archived**: These paywalls are no longer in use and have been archived. You can always restore the archived paywall to an active state.

---

title: "Create paywall" description: ""

metadataTitle: ""

import Details from '@site/src/components/Details';

A paywall serves as an in-app storefront where customers can browse products and make purchases.

▶ Before you start creating paywalls (Click to Expand)

1. Create at least one product.
2. (optional) Create offer.

To create a new paywall in the Adapty dashboard:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it. The paywall list page in the Adapy dashboard provides an overview of all the paywalls that have been set up in your account along with their metrics.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **Create paywall** button.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened **Paywalls/ New paywall** page, enter the ***Paywall name:** *The name that identifies the paywall throughout the Adapty Dashboard.
4. Click the **Add product** button.
5. From the opened drop-down list, select the **Products** that will be shown to your customers. The list contains only previously created products. The order of the products is preserved on the SDK side, so it's important to consider the desired order when configuring the paywall.
   Please note that after your paywall will get shown on the production at least once it will be impossible to change the products on the paywall as this may affect the paywall metrics.

./productto_paywall.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. If you serve your products with a free trial and other offers, consider adding them here, otherwise, they won't work. For this, click the **Add offer** button next to the product the offer belongs to. The button is active only if offers are set up for the product.
   Apple's intro offers get sorted out automatically, so you don't have to add them separately in Adapty.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

7. Click the **Create as a draft** button to confirm the paywall creation.

Now the paywall is created and you can add it to a placement.

---

title: "Design paywalls with legacy Paywall Builder" description: "Craft tailored in-app storefronts effortlessly using Adapty's Paywall Builder. No coding or design expertise required. Customize appearance, messaging, and buttons in real time. Elevate sales, highlight content, or provide exclusive features with ease"

metadataTitle: "Design Custom Paywalls Seamlessly with Adapty Paywall Builder"

Paywall Builder is a simple no-code tool that lets you create custom paywalls â€" in-app storefronts in your mobile app where users can make purchases. It doesn't require any technical or design skills. You can easily improve the appearance of your paywalls, change the messages they display, and add buttons as needed. Plus, you can make changes to these screens in real-time while your app is live, without releasing a new mobile app version.

The information below refers to the legacy Paywall Builder. For the new Paywall Builder, see New Paywall Builder.

Whether you're aiming to boost sales, highlight content, or offer exclusive features, the paywall builder offers an easy way to achieve your goals.

To use the Adapty Paywall Builder:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it.
2. Switch on the **Builder** toggle in the **Paywalls/ Your paywall** window.
3. You'll find a selection of paywall templates designed by professionals, ready for you to pick the one that best suits your needs. From there, you can make minor adjustments to tailor it exactly to your preferences.

onpaywall_builder.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

By leveraging the Adapty Paywall Builder, you'll be able to create persuasive paywalls that seamlessly align with your app's branding and purpose. For more details on customizing templates, layout, products, text, and buttons, as well as localizing them, refer to the following sections:

1. Paywall builder templates
2. Paywall layout and products
3. Paywall texts and buttons
4. Paywall fonts
5. Paywall custom tags
6. Paywall builder localization

After you configure your paywalls and add them to placements, you can show them in your mobile app. Refer to the Display Paywall Builder paywalls topic for details on how to do it.

---

title: "Paywall builder templates" description: ""

metadataTitle: ""

Adapty paywall builder simplifies the process of creating paywallsâ€”specialized screens within your app where users can make purchases. This tool eliminates the need for technical expertise or design skills. You can effortlessly customize how your paywalls look, the messages they convey, and where essential buttons are positioned. What's more, you can even make real-time changes to these screens while your app is running â€” without App Store/Google Play reviews.

Moreover, Adapty empowers you to optimize your paywalls further with A/B testing. Alongside the paywall builder, this allows you to test different variations of your paywalls to find the most effective design and messaging. Whether you're striving to increase sales, promote content, or grant access to exclusive features, the paywall builder provides a user-friendly solution to accomplish these objectives.

When creating or editing paywalls using Adapty's paywall builder, you have the flexibility to choose from three distinct layout options, each with its own corresponding templates. The template layout you select will dictate the visual appearance and user experience of the paywall in your iOS or Android app. Let's explore the three available template layout options:

Overlay

The overlay templates offer a versatile and engaging design for your paywalls. It consists of two layers, combining an image at the bottom and a layer showcasing your products or content on top. This approach ensures a visually appealing presentation that captures users' attention. On smaller screens, the layer containing products or content will overlay the image while scrolling, emphasizing the key elements of your paywall. This layered arrangement enhances user interaction and allows you to present multiple offers seamlessly.

2023-09-05at_15.48.442x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Transparent

The transparent layout is ideal for scenarios where you have a concise selection of products or content to present. This layout type offers a captivating visual experience by featuring a full-screen image that immediately grabs users' attention. With no scrolling involved, the image takes up the entire screen space, allowing you to make a bold statement and showcase your offerings in a straightforward manner.

2023-09-05at_15.49.422x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Note: Since there is no scrolling in the Transparent Layout, it is best suited for situations where you have a limited amount of content to display.

Flat

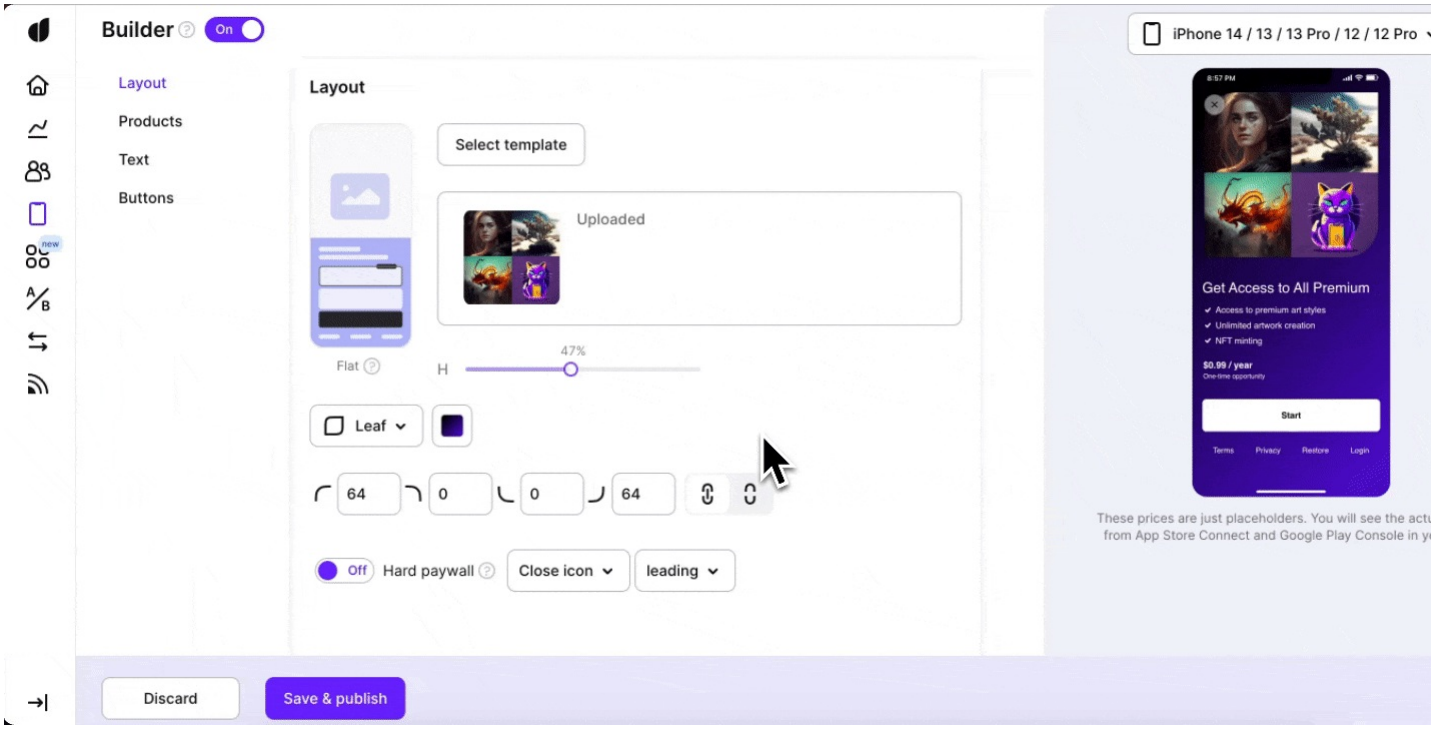
The flat layout is reminiscent of a streamlined landing page, presenting all the essential elements in a single continuous layer. Users can scroll through the content seamlessly, experiencing a fluid narrative as they engage with your paywall. This layout type offers a cohesive storytelling approach, enabling you to present your products, services, or content in a compelling and unified manner.

2023-09-05at_15.52.312x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Tip: The flat layout is particularly effective when you have a narrative or sequence of offerings to present to your users.

Device compatibility preview

Use the drop-down menu above the asset to select different devices, providing a preview of how your layout will appear on various screens. This feature enables you to ensure that your paywall looks optimal across different devices and screen sizes.



---

title: "Paywall layout and products" description: ""

metadataTitle: ""

Adapty paywall builder simplifies the process of creating paywallsâ€”specialized screens within your app where users can make purchases. This tool eliminates the need for technical expertise or design skills. You can effortlessly customize how your paywalls look, the messages they convey, and where essential buttons are positioned. What's more, you can even make real-time changes to these screens while your app is running â€” without App Store/Google Play reviews.

Moreover, Adapty empowers you to optimize your paywalls further with A/B testing. Alongside the paywall builder, this allows you to test different variations of your paywalls to find the most effective design and

messaging. Whether you're striving to increase sales, promote content, or grant access to exclusive features, the paywall builder provides a user-friendly solution to accomplish these objectives.

In this section, we will discuss the customization of the layout and products of your paywalls.

Layout

After selecting the preferred layout type and corresponding template for your paywall in Adapty's paywall builder, you gain the ability to shape the visual appearance of your paywall, making it engaging and aligned with your brand's aesthetics. This tab offers a range of controls that allow you to customize various aspects of the paywall's layout, background, and appearance. Let's explore the controls and options available in the Layout tab:

Main image and sizing

The **main image** is the centerpiece of your paywall's design, influencing the overall look and feel. This image serves to captivate users and encourage them to take action. Here are some guidelines to consider when selecting and uploading your main image:

- A photo should be PNG or JPEG < 2Mb.
- Photos with the main object in the center and some free space around it usually make the message clear.
- Emotional and/or bright photos work.
- Graphic works but use it without claims as there is a separate place for texts in the asset.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You have the ability to control the sizing of the main image, determining its proportions in relation to the entire paywall screen. This helps in achieving the desired visual balance and impact. Specify the image sizing as a percentage of the total screen area.

Mask type for the image

The mask type determines the shape of the main image, allowing you to apply creative effects to the visual presentation. Choose from the following mask types:

- Rectangle
- Rounded Rectangle
- Circle
- Leaf

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can adjust the roundness of the image mask using numerical values (not available for circle type).

Background color

The background color of the paywall sets the tone for the entire experience. You can choose a background color that aligns with your brand's identity or complements the image. The background color acts as a canvas that enhances the visual appeal of the paywall. You have the option to select either a solid color or a gradient color for the background.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Font settings of your paywall

It's important to keep your paywall visually consistent with the rest of your app â€” and one of the biggest visual factors is the font that you're using. You can choose to simply have a system font for your paywall (SF Pro for iOS, Roboto for Android), use one of the available common fonts or upload your own custom font:

2024-01-12at_18.47.472x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note Changing the font settings in Layout affects all the other labels on the paywall (unless they have been customised before that).

You can learn how to upload your custom font here. :::

Hard and soft paywalls

A key decision you can make in the layout tab is whether to enable a closing button, resulting in a soft paywall, or to remove it, resulting in a hard paywall. By toggling the ***Hard paywall** *option, you can instantly see how the closing button adapts or disappears, based on your choice.

For soft paywalls, you can set up the view and behavior of the closing paywall button.

Close button, its style, placement and fade-in animation

The presence of the closing icon provides users with the means to dismiss the paywall and continue their interaction with the app.

For soft paywalls, you can define the view of the closing paywall button and how fast it will appear:

1. Switch off the **Hard paywall** toggle.
2. In the expanded section, pick how the button should look and where it should be. The preview on the right will instantly change to reflect your choice.

buttondefinition.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

- **Close icon type:** Choose the icon for the Close button or **Custom text** for text buttons.
- **Icon placement:** Position the Close button either at the top-left, center or top-right part of the screen.
- **Color and opacity:** You can control the color and opacity of both the content and the background of the closing button. You can make the closing button fit your paywall better by adjusting the colors and/or removing the background of a button entirely.

3. To add a delay before showing this Close button, switch on the **Show after delay** toggle.

afterdelay.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. In the expanded section, specify the delay duration (in milliseconds) before the Close button starts fading in and indicate how long the button's fade-in animation should last.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Here is a video to show the whole process:

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Device compatibility preview

Use the drop-down menu above the asset to select different devices, providing a preview of how your layout will appear on various screens. This feature enables you to ensure that your paywall looks optimal across different devices and screen sizes.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Products

In the products tab of Adapty's paywall builder, you have the ability to create and customize a visually appealing product section that showcases your offerings to users. This tab enables you to configure various aspects of the product's appearance and textual content. Let's delve into the options available for customization in the products tab:

productssection.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

1. Product section layout

You can define how your product section is presented to users by choosing between a Horizontal List or a Vertical List layout. This layout selection influences how your products are arranged on the screen.

2. Main product

Highlighting a specific product can draw user attention. In the main product configuration, you can select the product that will be emphasized and preselected in the product section by a special border. This can be particularly useful for promoting a featured item or a special offer. You can also add badge text to this product to provide additional context or highlight its uniqueness â€” see how below.

3. Products customization

Let's take a look at how you can customize each product on your paywall:

(1, 2) You can adjust the color and the corner radius of the product containers for a unique visual touch. And you can also have a badge for every product on a paywall to provide additional context such as the savings for the user or highlight its uniqueness.

(3) You can also control the font for each of the text labels â€” bold, italic or regular as well as upload your own custom fonts (learn more about it here):

(4) But what's probably more important is that you can describe your products in text and fine-tune text attributes such as size, color and style. You can also use optional second title and subtitle to provide more details â€” for example, to let people compare the pricing between products more easily. This level of control allows you to craft a visually appealing and informative product display within your paywall, optimizing the user experience.

:::note Using tag variables

Every text field in the product block supports tag variables such as `<PROD_TITLE/>` for the title of the product. We strongly recommend using them for easier localization. :::

(5) If you intend to use this paywall in cases where a user might be eligible for an offer, you should configure the offer text for every expected type. And of course you can adjust its color and size as well. If a user is found eligible â€” the corresponding offer text will be shown as a subtitle on the product card.

:::warning Default text for offers

Note: eligibility for an offer is determined on a device by SDK. In case you haven't provided a custom text for it and the user was found eligible â€” our SDK will show a default text corresponding to the offer type.

Learn more about offers here. :::

Product styles: synced or separate

By default changing each of the style components above (such as any color, font size, or corner radius) applies to all of the products on a paywall. But if you want to make a particular product pop, you can disable the Style sync for this product and tune its visuals separately. This can be especially useful when highlighting the main product.

You can disable Style sync for a product in the upper right corner. After that, any changes you make to the visuals will only be applied to this product:

You can use this control to style any one of your products separately from the rest. But remember: when you turn style sync back on after making changes â€” this will revert them, so be careful.

Style sync also doesn't affect texts as those are always separate for every product.

Preview products on your Paywall

Once you've finished customizing how your products look, it makes sense to double-check the result using preview before testing it on a device. You can find some useful settings for it in the "eye" icon in the top-right corner:

There you can toggle whether you'd like to see our placeholders for the tag variables or the actual values of the tags. Or you can simulate what the paywall will look like if a user is found eligible for a certain offer type.

:::warning Preview uses placeholder values for tag variables

Remember: if you use tag variables for your products, the Preview only shows some hardcoded placeholder values for them. This is applied both to prices and titles for both the products and offers. The actual values will be retrieved from the App Store and Google Play and shown only on the device. :::

---

title: "Tag variables for product info in Paywall builder" description: ""

metadataTitle: ""

Adapty paywall builder has a way to customize all the text corresponding to your products and their offers. But if you have multiple locales â€” we strongly recommend using variables.

How it works

When texts for your products contain a tag variable from our list, our SDK uses the pre-fetched localized data from the stores to put it in place of a tag. That way the text on your paywall is always tailored for the right locale.

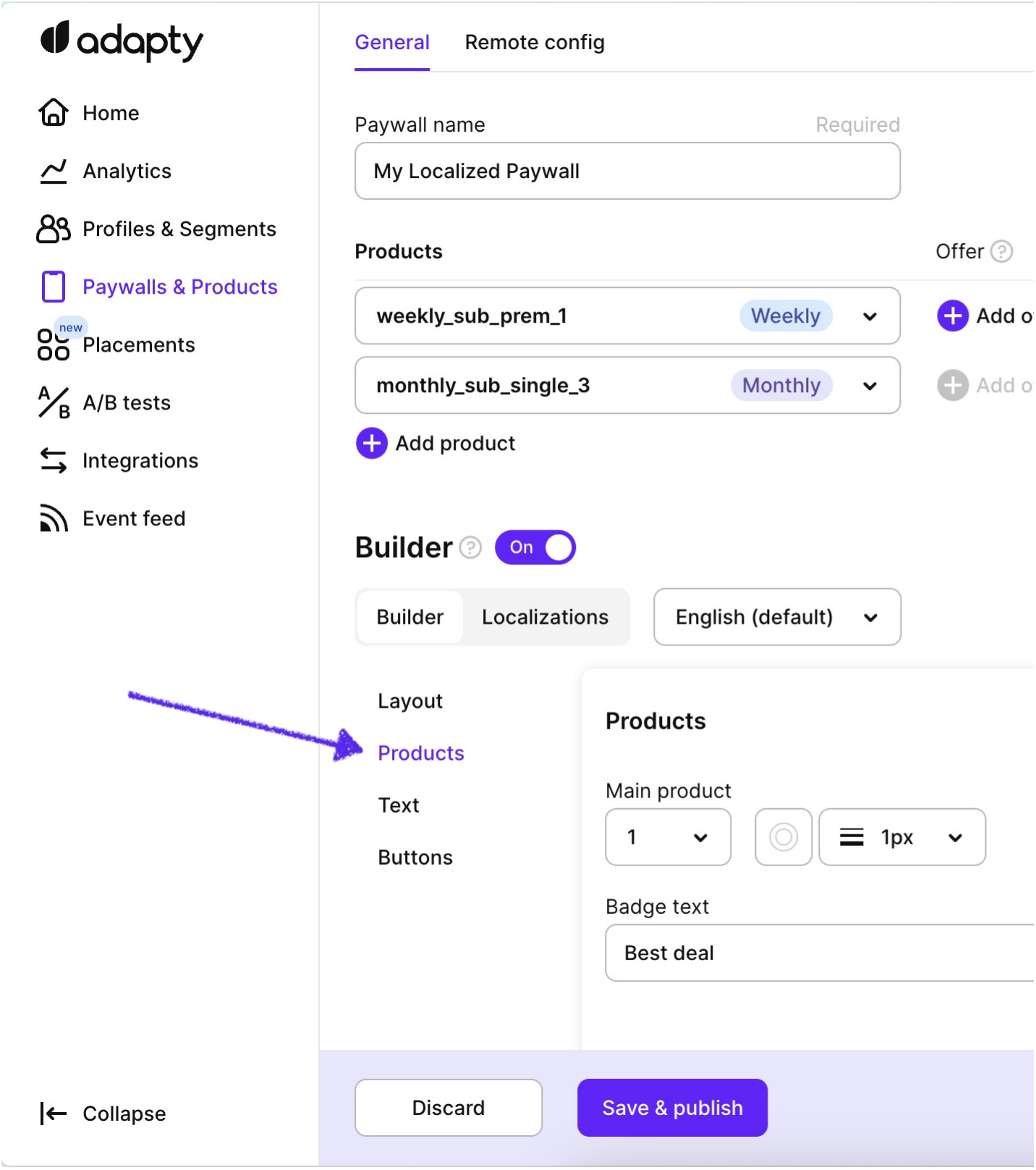
Example: suppose you have a "Premium Subscription" and your app is available both in the US and Spain. So you're selling "Premium Subscription for $4.99/month" in the US and "SuscripciÃ³n Premium por â‚¬4.99/mes" in Spain.

Tag variables allow you to rely on the data obtained directly from the store to localize such strings â€” so titles and prices will always be correct.

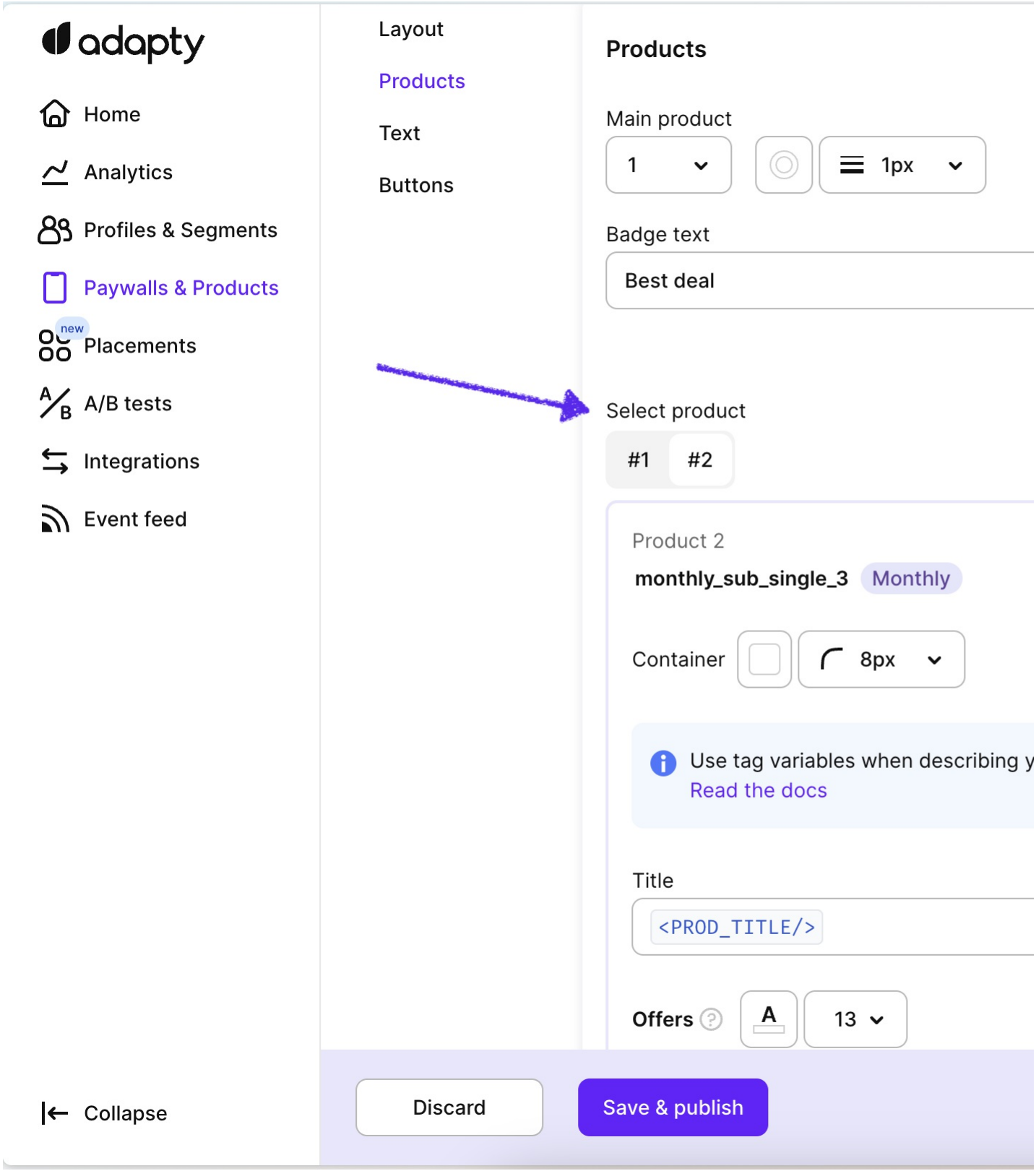
How to use tag variables

:::note You can only use tag variables when describing products and offers in the "Products" tab of the Paywall Builder :::

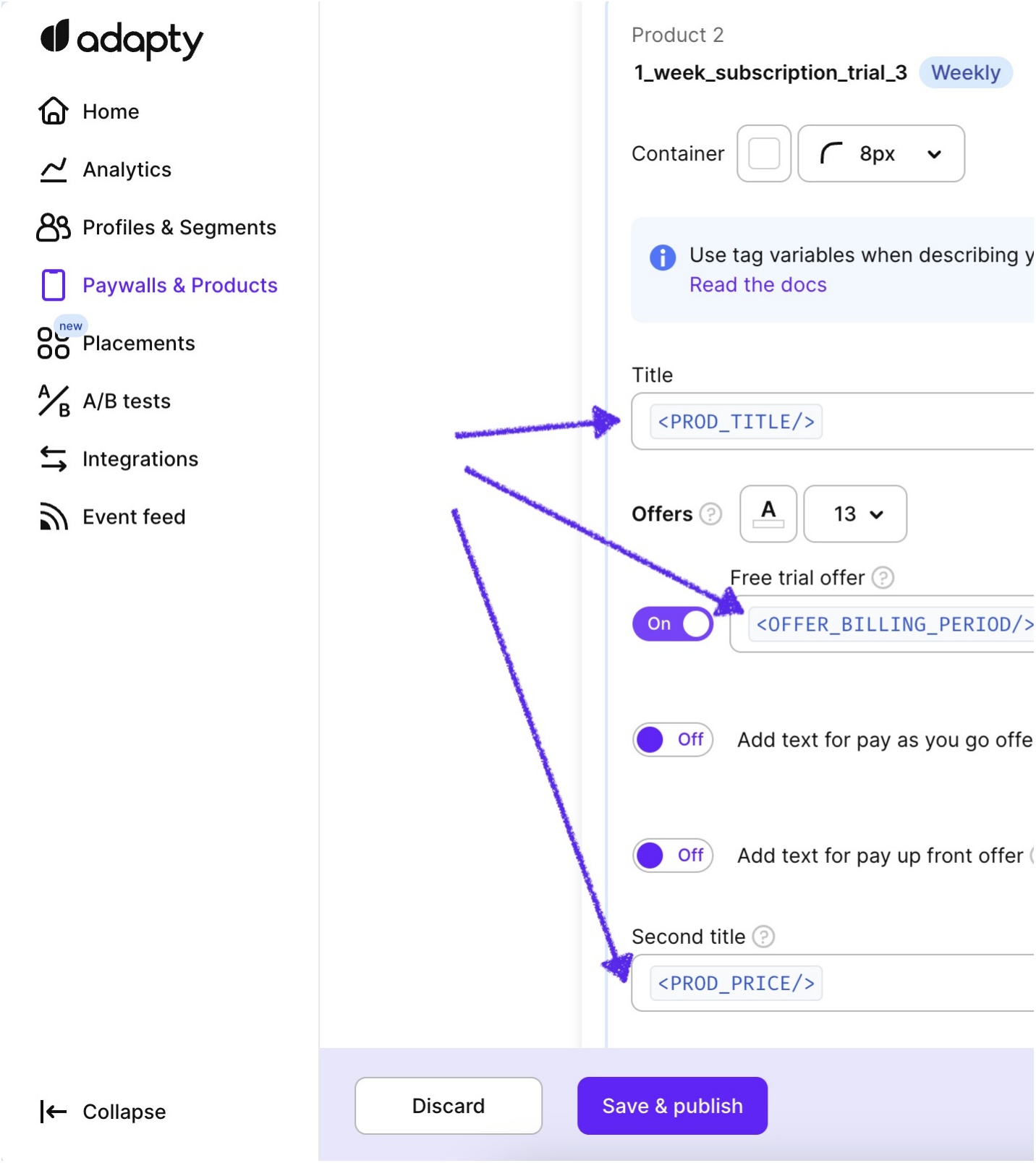
1. Select the "Products" tab of the Builder



2. Choose the product you'd like to customize:



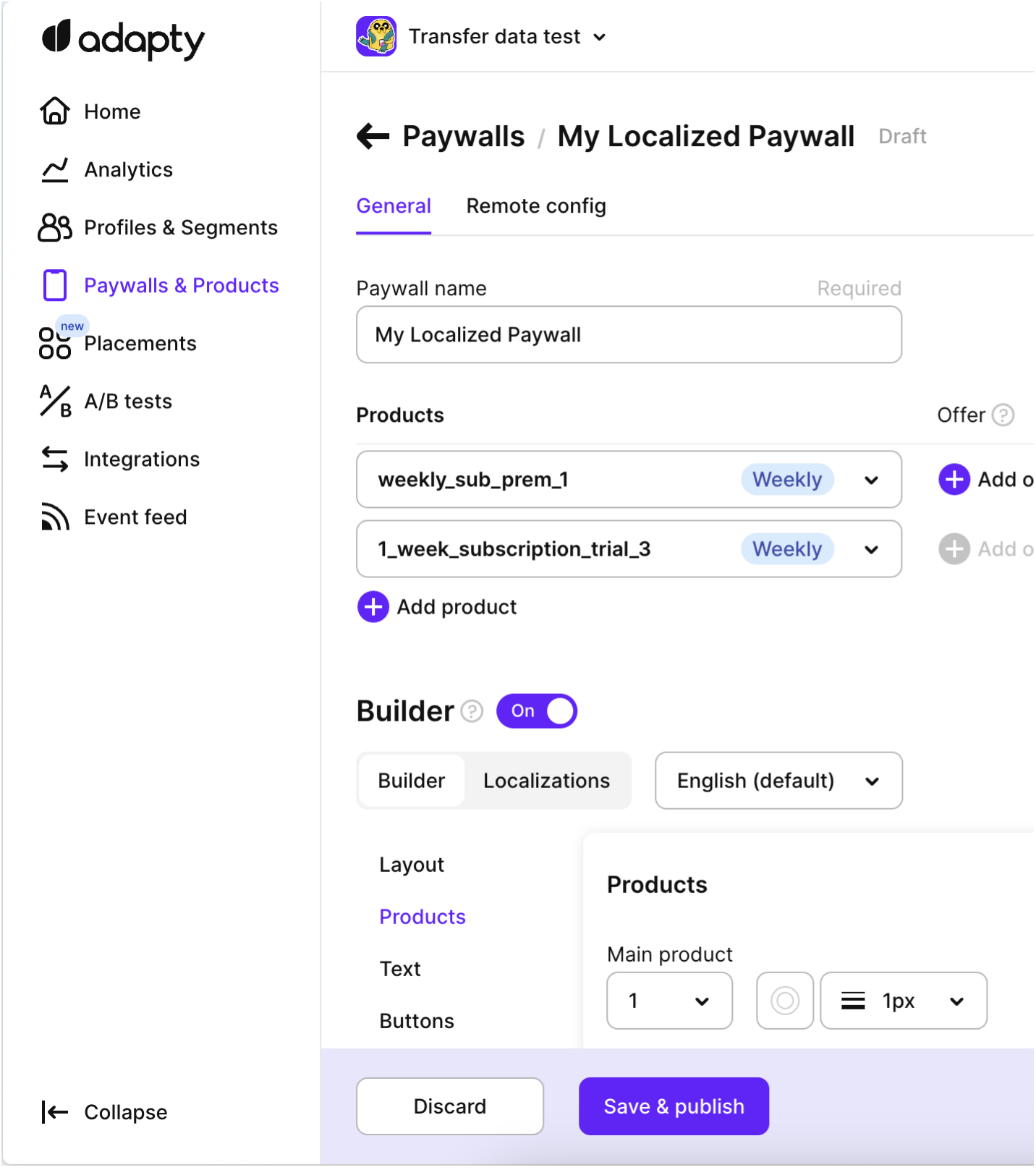
3. Use variables from the table below in any of the text fields to describe the product and its offers:



4. Check Preview on the right side of the screen to make sure all renders as intended.

:::warning **Note:** Preview doesn't use any real values to put in place of your variables as they are only obtained on a device by our SDK. However, by default it displays some template data that's in the same format as the real result.

You can disable this behavior by pressing "eye" icon in the top-right corner of the Preview and switching off the **"Tags preview values"** switch. The preview will then show the actual values of the variables:



:::

Full list of variables

| Tag variable | Description | Example | | :---------- | :---------- | :---------- | | `<PROD_TITLE/>` | Localized title of the product | Premium Subscription | | `<PROD_PRICE/>` | Localized price of the product. In case of subscriptions the price is for one billing period. | $9.99 | | `<PROD_PRICE_PER_DAY/>` | Price of a subscription divided by the number of days in the billing period. **Returns empty string for non-subscriptions.** | $0.33 | | `<PROD_PRICE_PER_WEEK/>` | Price of a subscription divided by the number of weeks in the billing period. **Returns empty string for non-subscriptions.** | $2.33 | | `<PROD_PRICE_PER_MONTH/>` | Price of a subscription divided by the number of months in the billing period. If actual billing period is less than a month â€" it's multiplied to represent how much a user would pay for a full month.**Returns empty string for non-subscriptions.** | $9.99 | | `<PROD_PRICE_PER_YEAR/>` | Price of a subscription that represents how much a user would pay for a full year of usage. So, for example, the price for the monthly subscription would be multiplied by 12 and the price for the yearly one would remain the same. **Returns empty string for non-subscriptions.** | $119.88 | | `<OFFER_PRICE/>` | Localized price of an offer (intro or promo). **Applicable only to auto-renewable subscriptions, returns empty string if user is not eligible for any offers** | $0.99 | | `<OFFER_BILLING_PERIOD/>` | Localized billing period of an offer (intro or promo). Same as `<OFFER_FULL_DURATION/>`for trial and pay-upfront offers. **Applicable only to auto-renewable subscriptions, returns empty string if user is not eligible for any offers** | 1 week | | `<OFFER_FULL_DURATION/>` | Localized full duration of an offer (intro or promo). **Applicable only to auto-renewable subscriptions, returns empty string if user is not eligible for any offers** | 1 month |

Offer tags for different offer types

:::note You can learn more about Offers and how you can configure them in Adapty here :::

Offer tags for different offer types might be confusing, so let's consider an example. Suppose we have a weekly subscription called "Premium Subscription" for a price of $5. For it we have 3 possible offers:

- **Pay As You Go**. First 3 weeks for a price of $3 (billed each week), then $5/week
- **Pay Up Front**. First 3 weeks for a price of $8 (billed right now), then $5/week
- **Free Trial**. First week free, then $5/week.

`<PROD_TITLE/>` for this product would be "Premium Subscription" and its `<PROD_PRICE/>` would be $5. But the values for the offer tags â€" depending on which offer the user is eligible for â€" would be:

| Tag variable | Pay As You Go | Pay Upfront | Free Trial | | :---------- | :---------- | :---------- | :--------- | | `<OFFER_PRICE/>` | $3 | $8 | $0 | | `<OFFER_BILLING_PERIOD/>` | 1 week | 3 weeks | 1 week | | `<OFFER_FULL_DURATION/>` | 3 weeks | 3 weeks | 1 week |

So in any offer other than a "Pay As You Go" type, `<OFFER_BILLING_PERIOD/>` and `<OFFER_FULL_DURATION/>`would be the same.

---

title: "Paywall texts and buttons" description: ""

metadataTitle: ""

Adapty paywall builder simplifies the process of creating paywallsâ€”specialized screens within your app where users can make purchases. This tool eliminates the need for technical expertise or design skills. You can effortlessly customize how your paywalls look, the messages they convey, and where essential buttons are positioned. What's more, you can even make real-time changes to these screens while your app is running â€” without App Store/Google Play reviews.

Moreover, Adapty empowers you to optimize your paywalls further with A/B testing. Alongside the paywall builder, this allows you to test different variations of your paywalls to find the most effective design and messaging. Whether you're striving to increase sales, promote content, or grant access to exclusive features, the paywall builder provides a user-friendly solution to accomplish these objectives.

In this section, we will discuss the customization of buttons and text elements within your paywalls.

Buttons

In the buttons tab, you have the ability to define and customize various buttons that play a crucial role in guiding user interactions and enhancing the overall user experience of your paywall. This tab empowers you to configure primary Call to action (CTA) buttons as well as secondary buttons that can direct users to essential legal information. You can enable or disable the appearance of the secondary buttons. Let's deep dive into the options available for customization in the buttons tab:

2023-09-01at_16.46.492x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

Primary call to action button

The primary call to action button serves as the key action that you want users to take. You can tailor its appearance and text to align with your paywall's goals:

- Button text: Define the text that appears on the primary call to action button.
- Text font size: Adjust the font size of the button text to ensure optimal readability.
- Button color: Choose a color that stands out and draws users' attention.
- Button roundness: Modify the roundness of the button's corners for a unique aesthetic.
- Button text color: Select a text color that complements the button's background and enhances legibility.

Secondary buttons

In addition to the primary call to action button, you can include secondary buttons that direct users to essential legal information. These buttons typically appear at the bottom of the paywall and provide users with access to important resources such as terms and privacy of the app usage. You can configure each secondary button with the following settings such as text, text size, text color, and URLs.

Apart from the traditional button configurations, the Restore and Login buttons provide specific functionalities:

- Restore: This button is used to allow users to restore their previous purchases or access content they've previously owned.
- Login: The login button facilitates user authentication and access to personalized content.

Texts

./ width: '700px', / image width */ display: 'block',* / for alignment */ margin: '0 auto'* / center alignment */ }} />

In the texts tab, you can make your paywall sound attractive and clear. This tab helps you tell users why your paywall is awesome. Just choose your words, make them look nice, and guide users through the cool stuff they'll get.

Feel free to use custom tags to personalize your UI text and custom fonts to make your paywall blend in more with the rest of your app's design.

Here are the main elements of the tab:

Headline and subhead

Make a catchy title and a small introduction that sets the mood. Keep it short and interesting.

Main features

Under the headline and subhead, show what cool stuff your paywall offers. You can do this in two ways:

- Feature list: Make a list of cool things users get with your subscription. You can add icons to show what each feature is about.
- Timeline: Show how things get better over time. Give each step a title, a small description, and an icon.
  For all of these elements you can control the alignment with the page, text size, and color individually.

---

title: "Custom fonts in paywall builder" description: ""

metadataTitle: ""

:::note Custom fonts are only available on AdaptyUI SDK v.2.1.0 and higher :::

One of the hallmarks of great design is consistency in visuals. So when building no-code paywalls you might want to make sure you use a custom font for your paywall to match the rest of your app. Here we'll talk about how we work with customising fonts and how you can use them.

What can be customized

Every text you see in Paywall Builder can have its own font and style. You can adjusted this in font controls for every text element:

2024-02-07at_13.27.092x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

But in some cases, it'd be more convenient to change the font on the entire paywall. This can be done in the Layout section of Paywall Builder by adjusting the Paywall Font.

Fonts available by default

When you create a paywall in Builder, Adapty uses a system font by default. That usually means SF Pro on iOS and Roboto on Android (though it can vary depending on the device). You can also pick one of the fonts commonly used across the apps (Arial, Times New Roman, Courier New, Georgia, Palatino and Verdana). There are also a few styles to choose from available for each of those fonts:

2024-01-12at_19.33.072x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

:::note Note that these fonts are not supplied as part of Adapty SDK and are only used in preview purposes. We can not guarantee they will work perfectly well on all of the devices.

However in our testing we observed that those fonts are typically recognised by most devices without additional effort from your side. You can also checkout which fonts are available by default on iOS here: In Apple official documentation :::

How to add a custom font to the Adapty Dashboard

If you need more than what is offered by default, you will need to add a custom font. Click on "Add custom font" in any of the font dropdowns and you'll see this screen:

2024-02-07at_13.21.552x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

From here you need to:

- Locate your font file and upload it to this form
- Provide a name to reference it in the Paywall Builder
- Specify the correct font names for both platforms
- Add the font file to your app's bundle if you haven't done it already

:::warning We will not be sending the font file you upload to the device, it is only needed for preview purposes. Our SDK only receives the strings referencing the font to use it while rendering the paywall. So you have to make sure you include the same font file in the bundle and provide the correct platform-specific font names for everything to work smoothly. Don't worry, it's not going to take a lot of time. :::

:::note By uploading the font file to Adapty, you're also confirming that you have the right to use it in your app :::

Getting the correct font name on iOS

There are two ways to get the correct ID for a font: first involves some basic coding, second involves an app called "Font Book" available on mac OS.

If you've already added a custom font to your app's bundle â€" chances are you're already referencing it by the font name. To make sure simply call `UIFont.familyNames()` to get the family name of the font and then plug it into `UIFont.fontNames(forFamilyName: familyName)`. You can do this in `viewDidLoad` and then remove this code snippet:

```
swift title="Swift" override func viewDidLoad() { super.viewDidLoad() ... for family in UIFont.familyNames.sorted() { print("Family: \(family)") let names =
UIFont.fontNames(forFamilyName: family) for fontName in names { print("- \(fontName)") } } }
```

The `fontName` in the above snippet is exactly what we're looking for. It could look something like "MyFont-Regular"

Second method is way simpler: you just need to install the font on your Mac computer, open the Font Book app, find the font and use it's `PostScript name`:

2024-01-12at_20.32.222x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Getting the correct font name on Android

If you have properly added the font file to the resource folder, you simply need to provide the name of the file. Make sure it is lowercase and only contains letters, numbers and underscores â€" otherwise it might not work.

You can confirm that the filename is correct by calling `ResourcesCompat.getFont(context, R.font.my_font)` with `my_font` being the filename you're using. In this case `my_font` is exactly what you should put in while creating a custom font in Adapty.

Adding the font files to your app's bundle

Chances are you're already using a custom font in other parts of your app. But if you don't â€" make sure to include the font file in your app's project and bundle. Read how to do it below.

On iOS: In Apple official documentation

On Android: In Android official documentation

---

title: "Custom tags in paywall builder" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

:::note Custom tags are only available on AdaptyUI SDK v.2.1.0 and higher :::

Custom tags are a feature designed to avoid creating separate paywalls for different situations. Imagine having a single paywall that adapts to different scenarios by incorporating specific user data. For instance, a simple greeting like "Hello!" can transform into a personalized message, such as "Hello, John!" or "Hello, Ann!"

Various ways to use:

- Userâ€™s email/name on the paywall
- Current day of the week on the paywall to increase sales (as in â€œHappy Thursdayâ€œ)
- Custom properties of the products you're selling (name of the personalized fitness program, phone number in the VoIP app, etc)

Custom tags enable you to create a consistent paywall for various situations, allowing your app's user interface to dynamically incorporate the relevant information. It's a practical solution for tailoring a paywall design for each specific user.

:::warning Make sure to add fallbacks for every line with custom tags

In some cases your app might not know what to replace a custom tag with: for example, if your Paywall is delivered to users on the older versions of AdaptyUI SDK.

So when using custom tags, make sure to add fallback lines â€" they will be used to replace the lines containing unknown custom tags. Otherwise the user will see custom tags as code (`<USERNAME/>`). :::

How to add a custom tag to a paywall

Every text line you see in Paywall Builder can have one or more custom tags.

To add a custom tag to a line:

1. Enter the custom tag you want in the format `<CUSTOM_TAG/>` or simply type an opening angle bracket (<) in the text line followed by the custom tag you need. The system will then offer you the tag in the correct format.

   Please pay attention that:

   - In the Adapty paywall builder, custom tags are wrapped in angle brackets (`<CUSTOM_TAG/>`) while in mobile app code, you should refer to them directly (`CUSTOM_TAG`).
   - Custom tags are case-sensitive.
   - Custom tags can't overlap with any of the Tag Variables reserved for product info in Adapty.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. After entering the custom tag, make sure to enter the fallback line. The fallback is the text displayed in your app if it does not know about a particular custom tag. This ensures that users won't see the custom tag as code; instead, they'll see the designated fallback text. Please note that the fallback replaces the entire line containing the custom tag

forcustom_tag.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

How to use custom tags in your mobile app

To use custom tags in your mobile app, you need to create a `tagResolver` object. This is a dictionary/map containing custom tags and the string values to replace them with when rendering the paywall in your app. Here's an example:

```
swift title="Swift" let tagResolver = [ "USERNAME": "John", ] kotlin title="Kotlin" val customTags = mapOf("USERNAME" to "John") val tagResolver = AdaptyUiTagResolver { tag ->
customTags[tag] } java title="Java" Map<String, String> customTags = new HashMap<>(); customTags.put("USERNAME", "John"); AdaptyUiTagResolver tagResolver = customTags::get;
```

In this example, `USERNAME` is a custom tag that you entered in the Adapty dashboard while designing a paywall as `<USERNAME/>`. The `tagResolver` ensures that when your app encounters this custom tag, it dynamically replaces it with the specified value, in this case, `John`.

We recommend to create and populate the `tagResolver` right before presenting your paywall. Once it is created, pass it over to the AdaptyUI method used for presenting. Read more on how to present paywalls on iOS, Android, Flutter, React Native, or Unity.

---

title: "Design paywall with remote config" description: "Leverage the Paywall Remote Config tool to fine-tune your paywalls effortlessly. Utilize custom JSON payloads to personalize titles, images, fonts, and colors with precision. Ensure optimal performance with size restrictions per language, all without hassle"

metadataTitle: "Optimize Paywalls with Flexible Configuration using Paywall Remote Config"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

The Paywall Remote Config is a powerful tool that provides flexible configuration options. It allows the use of custom JSON payloads to tailor your paywalls precisely. With it, you can define various parameters such as titles, images, fonts, colors, and more, ensuring that the overall size remains within 10 KB per language.

▶ Before you start customizing a paywall (Click to Expand)

1. Create a product.
2. Create a paywall and add the product to it.

To start customizing a paywall using the remote config:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **3-dot** button next to the paywall and select the **Edit** option.

toremote_config.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened **Paywalls/ Your paywall** page, switch to the **Remote config** tab.

Remote config has 2 views:

- Table
- JSON

Both the **Table** and **JSON** views include the same configuration elements. The only distinction is a matter of preference, with the sole difference being that the table view offers a context menu, which can be helpful for correcting localization errors.
You can switch between views by clicking on the **Table** or **JSON** tab whenever necessary.

Whatever view you've chosen to customize your paywall, you can later access this data from SDK using the`remoteConfig` or `remoteConfigString` properties of `AdaptyPaywall`, and make some adjustments to your paywall. Here are some examples of how you can use a remote config.

```json { "screen_title": "Today only: Subscribe, and get 7 days for free!" }

# Test titles or others texts

</TabItem> <TabItem value="Images" label="Images" default> json { "background_image": "https://adapty.io/media/paywalls/bg1.png" }

# Test images on your paywall

</TabItem> <TabItem value="Fonts" label="Fonts" default> json { "font*family": "San Francisco", "font*size": 16 }

# Test fonts

</TabItem> <TabItem value="Color" label="Color" default> json { "subscribe*button*color": "purple" }

# Test colors of buttons, texts etc.

</TabItem> <TabItem value="HTML" label="HTML" default> json { "photo_gallery": "https://adapty.io/media/paywalls/link-to-html-snippet.html" }

# Any HTML code that can be displayed on the paywall

</TabItem> <TabItem value="Soft/Hard Paywall" label="Soft/Hard Paywall" default> json { "hard_paywall": true }

# By setting it to true, you disalow skipping paywall without subscribing

# You have to handle this logic in your app

</TabItem> <TabItem value="Translations" label="Translations" default> json { "title": { "en": "Try for free!", "es": "Â¡Prueba gratis!", "ru": "ÐŸÐ¾Ð¿Ñ€Ð¾Ð±Ñƒ Ð±ÐµÑÐ¿Ð»Ð°Ñ‚Ð½Ð¾!" } } ```

You can combine different options, and make up your own. This way you can test different titles, texts, images, fonts, colors, and so on.

### JSON view of the remote config

In the **JSON** view of the remote config, you can enter any JSON formatted data up to 10 kB per language:

configJSON.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

### Table view of the remote config

If it's not common for you to work with code and there is a need to correct some values of the JSON, Adapty has the **Table** view for you.

configtable.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

It is a copy of your JSON in the format of a table that is easy to read and understand. Color coding helps to recognize different data types.

To add a key, click the **Add raw** button. We automatically check the values and types mapping and show an alert if your corrections may lead to an invalid JSON.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Additional raw options are mostly useful for paywall localisations:

configtable_options.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Now it's time to create a placement and add the paywall to it. After that, you can display your remote config paywalls in your mobile app.

---

title: "Paywall localization" description: "Adapt your product for diverse markets with paywall localizations, offering versions in different languages to cater to specific regional needs. Learn how to add paywall locale using Adapty Paywall Builder or remote config"

## metadataTitle: "Enhance Global Reach: Implementing Paywall Localizations for Diverse Markets"

In a world with many cultures, it's important to adapt your product for each country. You can do this by using paywall localizations. For each paywall, you can make versions in different languages to match the needs of specific local markets.

Depending on what you use to design your paywalls, adding locale varies:

1. Adding paywall locale in Adapty Paywall Builder
2. Adding paywall locale in remote config

Once you add locales to a paywall, learn how to correctly work with locale codes in your app's code.

---

title: "Add paywall locale in Adapty Paywall Builder" description: "Effortlessly tailor your paywall for different markets by integrating locales within Adapty Paywall Builder. Learn how to enhance global reach and cater to specific regional needs"

## metadataTitle: "Optimize Localization: Adding Locale in Adapty Paywall Builder"

Localizing is a tedious process that requires time and precision. When using Paywall Builder, Adapty does almost all of the work for you, as most of the things you'll need work out of the box. This page describes how it works.

Suppose you've finished configuring your paywall in the default `en` localization and you like the result. Now it's time to add another language.

## Add and set up localization

1. Switch over to the **Localizations** tab below the **Builder** title.
```

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

2. Click the **Add locale** button, and select all languages you want to have in your app.

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

```
Once added, the new locale will be pre-filled with values from the default.
```

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

:::note Pay attention to the locale code (`en`, `fr` and `it` on the screenshot above). You'll need to pass it to the `getViewConfiguration` method of our SDK to get the correct localization.

You can learn more about it here. :::

3. Now you can fill in the translated values for the new locale. There are a few controls in this table that can make it easier (especially if you have many locales).

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

Feel free to use the context menu of the English locale to fix localization issues:

- Use the **Push this value to all locales** menu to overwrite any changes made in the row for non-English locales, replacing them with the values from the English locale.
- Use the **Revert all row changes to original values** menu to cancel all changes made in the current session, reverting them to the last saved values.

:::note Using tag variables

We strongly recommend using tag variables (such as `<PROD_TITLE/>`) to speed up your localization process and ensure that the text is always correct. Learn more about them here. :::

Preview the localization result

You can check your texts while editing by simply switching over back to the **Builder** tab and selecting another locale:

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

Once you add locales to a paywall, learn to correctly work with locale codes in your app's code.

---

title: "Add paywall locale in remote config" description: "Expand your paywall's reach across diverse markets by configuring locales in remote config. Learn how to optimize localization and cater to specific regional preferences"

metadataTitle: "Implement Multi-Language Support: Adding Paywall Locale in Remote Config"

In a world with many cultures, it's important to adapt your product for each country. You can do this by using paywall localizations. For each paywall, make versions in different languages â€”each one catering to a specific languageâ€”to match the needs of specific local markets.

If you've designed a paywall with the remote config, use the remote config as well to add localizations. Whether you're using a table view or JSON format, you can easily tweak settings for different languages. For example, you can translate String keys from English to Italian or set different Boolean values like TRUE for English and FALSE for Italian. You can even change background images based on the language. Basically, you keep the same setup but customize values for each language, making sure users get a personalized experience.

How to set up a localization for a paywall customized using remote config:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it and click the **3-dot** button next to the product and select the **Edit** option.

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

2. In the **Paywalls / Your paywall** window, switch to the **Remote config** tab.

toremote_config.png').default} style={{ border: '1px solid #727272', /* border width and color */width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

3. In the **Remote config** tab, click the **Add locale** button and select all languages you want to have in your app.

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

4. Click on the element you want to translate and enter a new value. You can both translate **String** and **List** values and replace pictures with those more appropriate for the locale.

configlocalization.png').default} style={{ border: '1px solid #727272', /* border width and color */width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

Feel free to use the context menu of the English locale to fix localization issues:

- Use the **Push this value to all locales** menu to overwrite any changes made in the row for non-English locales, replacing them with the values from the English locale.
- Use the **Revert all row changes to original values** menu to cancel all changes made in the current session, reverting them to the last saved values.

confiloc*table*options.png').default} style={{ border: '1px solid #727272', /* border width and color */width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

Once you add locales to a paywall, learn to correctly work with locale codes in your app's code.

---

title: "Fallback paywalls" description: ""

metadataTitle: ""

import Details from '@site/src/components/Details';

A paywall is an in-app storefront where customers can see and purchase products within your mobile app. Typically, paywalls are fetched from the server when a customer accesses them. However, Adapty allows you to have fallback paywalls for situations when a user opens the app without a connection to the Adapty backend (e.g., no internet connection or in the rare case of backend unavailability) and there's no cache on the device.

Adapty generates fallbacks as a JSON file in the necessary format, reflecting English versions of the paywalls you've configured in the Adapty Dashboard. To let your users see your fallback paywall:

1. Download the file from the Adapty Dashboard - one per app store and Adapty SDK version - as described below.
2. For iOS, Android, and React Native: Place it alongside your app on the user's device, and pass it to the `.setFallbackPaywalls` method. See detailed instructions for iOS, Android, and React Native.
3. For Flutter and Unity: Pass the contents of the file (the JSON string itself) to the `.setFallbackPaywalls` method. See detailed instructions for Flutter and Unity.

Download fallback paywalls as a file in the Adapty Dashboard

To integrate fallback paywalls into your mobile app code, start by downloading them from the Adapty Dashboard. The downloaded JSON file will contain one paywall for each placement, specifically the paywall designated for the `All users` audience in the Adapty Dashboard.

▶ Before you can download a paywall fallback (Click to Expand)

1. Create products you want to sell
   2. Create paywall and add the products to it.
   3. Create placement and add paywalls to it. Placement is the location where the paywall will be shown.

To download the JSON file with the fallback paywalls:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab or just the **Placements** section in the Adapty main menu.

./ width: '700px', / image width */display: 'block', /* for alignment */margin: '0 auto' */ center alignment */ }} />

2. In the **Products** or **Placements** window, click the **Fallbacks** button.
3. Select the platform for the fallbacks - **iOS** or **Android**.
4. In the **Download iOS/Android Fallback** window, select the SDK version you use for your app and click the **Download** button.

As a result, you will get a JSON file.

---

title: "Paywall metrics" description: ""

metadataTitle: ""

Adapty collects a series of metrics to help you better measure the performance of the paywalls. All metrics are updated in real-time, except for the views, which are updated once every several minutes. All metrics, except for the views, are attributed to the product within the paywall. This document outlines the metrics available, their definitions, and how they are calculated.

Paywall metrics are available on the paywall list, providing you with an overview of the performance of all your paywalls. This consolidated view presents aggregated metrics for each paywall, allowing you to assess their effectiveness and identify areas for improvement.

For a more granular analysis of each paywall, you can navigate to the paywall detail metrics. In this section, you will find comprehensive metrics specific to the selected paywall, offering deeper insights into its performance.

2023-07-19at_16.05.412x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Metrics controls

The system displays the metrics based on the selected time period and organizes them according to the left-side column parameter with three levels of indentation.

For Live paywall, the metrics cover the period from the paywall's start date until the current date. For inactive paywalls, the metrics encompass the entire period from the start date to the end of the selected time period. Draft and archived paywalls are included in the metrics table, but if there is no data available for those paywalls, they will be listed without any displayed metrics.

View options for metrics data

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

The paywall page offers two view options for metrics data: placement-based and audience-based.

In the placement-based view, metrics are grouped by placements associated with the paywall. This allows users to analyze metrics by different placements.

In the audience-based view, metrics are grouped by the target audience of the paywall. Users can assess metrics specific to different audience segments. You can select the preferred view using the dropdown option at the top of the paywall detail page.

Profile install date filtration

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

The Filter metrics by install date checkbox enables the filtering of metrics based on the profile install date, instead of the default filters that use trial/purchase date for transactions or view date for paywall views. By selecting this checkbox, you can focus on measuring user acquisition performance for a specific period by aligning metrics with the profile install date. This option is useful for customizing the metrics analysis according to your specific needs.

Time ranges

2023-07-19at_16.12.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

You can choose from a range of time periods to analyze metrics data, allowing you to focus on specific durations such as days, weeks, months, or custom date ranges.

Available filters and grouping

Adapty offers powerful tools for filtering and customizing metrics analysis to suit your needs. With Adapty's metrics page, you have access to various time ranges, grouping options, and filtering possibilities.

- âœ… Filter by: Audience, country, paywall, paywall state, paywall group, placement, country, store, product, and product store.
- âœ… Group by: Product and store.

You can find more information about the available controls, filters, grouping options, and how to use them for paywall analytics in this documentation.

Single metrics chart

One of the key components of the paywall metrics page is the chart section, which visually represents the selected metrics and facilitates easy analysis.

The chart section on the paywall metrics page includes a horizontal bar chart that visually represents the chosen metric values. Each bar in the chart corresponds to a metric value and is proportional in size, making it easy to understand the data at a glance. The horizontal line indicates the timeframe being analyzed, and the vertical column displays the numeric values of the metrics. The total value of all the metric values is displayed next to the chart.

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Additionally, clicking on the arrow icon in the top right corner of the chart section expands the view, displaying the selected metrics on the full line of the chart.

Total metrics summary

Next to the single metrics chart, the total metrics summary section is shown, which displays the cumulative values for the selected metrics at a specific point in time, with the ability for you to change the displayed metric using a dropdown menu.

2023-07-19at_16.19.332x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Metrics definitions

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Revenue

This metric represents the total amount of money generated in USD from purchases and renewals. Please note that the revenue calculation does not include the App Store / Play Store commission and is calculated before deducting any fees.

Proceeds

This metric represents the actual amount of money received by the app owner in USD from purchases and renewals after deducting the applicable App Store / Play Store commission. It reflects the net revenue that directly contributes to the app's earnings. For more information on how proceeds are calculated, you can refer to the Adapty documentation.

ARPPU

ARPPU is an average revenue per paying user. Itâ€™s calculated as total revenue divided by the number of unique paying users. $15000 revenue / 1000 paying users = $15 ARPPU.

ARPAS

The average revenue per active subscriber allows you to measure the average revenue generated per active subscriber. It is calculated by dividing the total revenue by the number of subscribers who have activated a trial or subscription. For example, if the total revenue is $5,000 and there are 1,000 subscribers, the ARPAS would be $5. This metric helps assess the average monetization potential per subscriber.

Unique conversion rate (CR) to purchases

The unique conversion rate to purchases is calculated by dividing the number of purchases by the number of unique views. For example, if there are 10 purchases and 100 unique views, the unique conversion rate to purchases would be 10%. This metric focuses on the ratio of purchases to the unique number of views, providing insights into the effectiveness of converting unique visitors into paying customers.

CR to purchases

The conversion rate to purchases is calculated by dividing the number of purchases by the total number of views. For example, if there are 10 purchases and 100 views, the conversion rate to purchases would be 10%. This metric indicates the percentage of views that result in purchases, providing insights into the effectiveness of your paywall in converting users into paying customers.

Unique CR to trials

The unique conversion rate to trials is calculated by dividing the number of trials started by the number of unique views. For example, if there are 30 trials started and 100 unique views, the unique conversion rate to trials would be 30%. This metric measures the percentage of unique views that result in trial activations, providing insights into the effectiveness of your paywall in converting unique visitors into trial users.

Purchases

Purchases represent the cumulative total of various transactions made on the paywall. The following transactions are included in this metric (renewals are not included):

- New purchases, that are made directly on the paywall.
- Trial conversions of trials that were initially activated on the paywall.
- Downgrades, upgrades, and cross-grades of subscriptions made on the paywall.
- Subscription restores on the paywall, such as when a subscription is reinstated after expiration without auto-renewal.

By considering these different types of transactions, the purchases metric provides a comprehensive view of the overall acquisition and monetization activity on your paywall.

Trials

The trials metric represents the total number of trials that have been activated. It reflects the number of users who have initiated trial periods through your paywall. This metric helps track the effectiveness of your trial offering and can provide insights into user engagement and conversion from trials to paid subscriptions.

Trials canceled

The trials canceled metric represents the number of trials in which the auto-renewal feature has been switched off. This occurs when users manually unsubscribe from the trial, indicating their decision not to continue with the subscription after the trial period ends. Tracking trials canceled provides valuable information about user behavior and allows you to understand the rate at which users opt out of the trial.

Refunds

The refunds metric represents the number of refunded purchases and subscriptions. This includes transactions that have been reversed or refunded due to various reasons, such as customer requests, payment issues, or any other applicable refund policies.

Refund rate

The refund rate is calculated by dividing the number of refunds by the number of first-time purchases (renewals are not included). For example, if there are 5 refunds and 1000 first-time purchases, the refund rate would be 0.5%.

Views

The views metric represents the total number of times the paywall has been viewed by users. Each time a user visits the paywall, it is counted as a separate view. For example, if a user visits the paywall two times, it will be recorded as two views. Tracking views helps you understand the level of engagement and user interaction with your paywall, providing insights into user behavior and the effectiveness of your paywall placement and design.

Unique views

The unique views metric represents the number of unique instances in which the paywall has been viewed by users. Unlike total views, which count each visit as a separate view, unique views count each user's visit to the paywall only once, regardless of how many times they access it. For example, if a user visits the paywall two times, it will be recorded as one unique view. Tracking unique views helps provide a more accurate measure of user engagement and the reach of your paywall, as it focuses on individual users rather than the total number of visits.

:::warning Make sure to send paywall views to Adapty using `.logShowPaywall()` method. Otherwise, paywall views will not be accounted for in the metrics and conversions will be irrelevant. :::

---

title: "Duplicate paywall" description: "Simplify the process of modifying existing paywalls in Adapty by duplicating them, preserving analytics integrity while making minor adjustments. Learn how to efficiently duplicate paywalls for seamless replacement as needed"

metadataTitle: "Streamlined Paywall Management: Duplicating Paywalls in Adapty"

If you need to make small changes to an existing paywall in Adapty, especially when it's already being used in your mobile app and you don't want to mess up your analytics, you can simply duplicate it. You can then use these duplicates to replace the original paywalls in some or all placements as needed.

This creates a copy of the paywall with all its details, like its name, products, and any promotions. The new paywall will have "Copy" added to its name so you can easily tell it apart from the original.

To duplicate a paywall in the Adapty dashboard:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it. The paywall list page in the Adapy dashboard provides an overview of all the paywalls present in your account.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **3-dot** button next to the paywall and select the **Duplicate** option.
3. Adjust the new paywall and click the **Save** button.
4. Adapty will prompt you to replace the original paywalls with their duplicates in placements if the original paywall is currently used in any placement. If you choose **Create and replace original**, the new paywalls will immediately go **Live**. Alternatively, you can create them as new paywalls in the **Draft** state and add them to placements later.

---

title: "Download fallback paywalls" description: "Explore how fallback paywalls ensure uninterrupted in-app purchases, even without an internet connection. Learn how Adapty's innovative solutions empower developers to define fallback paywalls for a seamless user experience."

metadataTitle: "Fallback Paywalls: Ensuring Seamless In-App Purchases | Adapty"

import Details from '@site/src/components/Details';

A paywall is an in-app storefront where customers can see and purchase products within your mobile app. Typically, paywalls are fetched from the server when a customer accesses them.

Adapty allows you to define fallback paywalls for the situations when a user opens the app without a connection to the Adapty backend (e.g., no internet connection or in the rare case of backend unavailability) and there's no cache on the device.

Adapty generates fallbacks as a JSON file in the necessary format, reflecting the default (English) versions of the paywalls you've configured in the Adapty Dashboard. Simply download the file - one per app store, place it alongside your app on the user's device, and pass its contents to the `.setFallbackPaywalls` method, following the instructions outlined below.

▶ Before you start adding local fallback paywalls (Click to Expand)

1. Create products you want to sell.
2. Create paywalls and add the products to the paywalls. Paywalls are in-app stores in your mobile apps.
3. Create placements and add paywalls to the placements. Placement is the location where the paywall will be shown.

To download the JSON file with the fallback paywalls:

1. Open the **Paywalls and Products** section from the Adapty main menu, then select the **Products** tab or just the **Placements** section in the Adapty main menu.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the **Products** or **Placements** window, click the **Fallbacks** button. In both cases, you will get the same file. Use its contents in the `fallback_paywalls` method in your mobile app code.

---

title: "Archive paywall" description: "Simplify your paywall management process in Adapty by archiving unused paywalls, ensuring they're safely stored for future access while decluttering your workspace. Learn how to efficiently archive paywalls and optimize your interface for improved efficiency."

metadataTitle: "Efficient Paywall Management: Archiving Unused Paywalls in Adapty"

As you dive into Adapty and fine-tune your paywall settings, you might accumulate paywalls that no longer fit your current strategy or campaigns. These unused paywalls, left `Inactive`, can clutter your workspace, making it tricky to find the ones that matter most. To tackle this, Adapty introduces the option to archive these unnecessary paywalls.

Archiving ensures they're safely stored without permanent deletion, ready to be accessed if needed in the future. Plus, archived paywalls can be filtered out from the default view, decluttering your workspace and simplifying your user interface. In this guide, we'll walk you through efficiently archiving paywalls in Adapty, giving you greater control over your paywall management process.

Just a friendly reminder: Live paywalls that are currently active in at least one placement cannot be archived. If you wish to archive such a paywall, simply remove it from all placements beforehand.

To archive a paywall:

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **3-dot** button next to the paywall and select the **Archive** option.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. When you're in the **Archive paywall** window, simply type in the name of the paywall you wish to archive and then click the **Archive** button.

---

title: "Return paywall from archive" description: "Optimize your paywall management process in Adapty by restoring archived paywalls, providing flexibility for reintroduction into your strategy when necessary. Learn how to efficiently return archived paywalls and enhance your paywall management experience"

metadataTitle: "Flexible Paywall Management: Returning Archived Paywalls in Adapty"

Having the ability to archive paywalls is a highly beneficial feature for streamlining your paywall management process. It allows you to conceal paywalls that are no longer needed, reducing clutter in your workspace. Moreover, the option to restore archived paywalls provides flexibility, enabling you to reintroduce them into your strategy if they prove to be useful again.

Archived paywalls may be filtered out of the default view. To see them, select **Archived** in the **State** filter.

To return a paywall back from the archive

1. Open the **Products and Paywalls** section in the Adapty main menu and click the **Paywall** tab to open it.
2. Make sure that archived paywalls are displayed in the list. If not, change the **State** filter.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. Click the **3-dot** button next to the archived paywall and select the **Back to active** option.

backto_active.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

---

title: "Placements" description: "Enhance user experience and maximize conversions by strategically deploying paywalls and A/B tests at various points in your app journey with Adapty's placement system. Learn how to manage, modify, and analyze placements effortlessly for improved engagement"

metadataTitle: "Optimize Engagement: Utilizing Adapty's Placements for Paywalls and A/B Tests"

With Adapty's placement system, you can create and run paywalls and A/B tests at different points in your app user's journey, such as Onboarding, Settings, etc. These points are called Placements.

A placement in your app can manage multiple paywalls or A/B tests at a time, each made for a certain group of users, which we call Audiences. Moreover, you can experiment with paywalls, replacing one with another over time without releasing a new app version.

The only thing you hardcode in the mobile app is the placement ID.

To see the list of placements in Adapty, open the **Placements** section in the Adapty main menu.

The **Placements** list offers a comprehensive view of various locations in the user journey where paywalls or A/B tests can appear. Each item in the list corresponds to a specific placement, allowing easy management and modification. You can edit placement details, associate them with the desired paywall or A/B test for a specified audience, or remove unnecessary placements. The numbers in the table reflect the analytics for placements since their activation.

2023-07-26at_14.51.342x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

From here you can:

- Create a new placement
- Edit an existing placement
- Delete an existing placement
- Download local fallback paywalls, those are the paywalls that will be used when a user opens the app and there's no connection with Adapty backend (e.g. no internet connection or in the rare case when the backend is down) and there's no cache on the device.

---

title: "Placements" description: "Enhance user experience and maximize conversions by strategically deploying paywalls and A/B tests at various points in your app journey with Adapty's placement system. Learn how to manage, modify, and analyze placements effortlessly for improved engagement"

metadataTitle: "Optimize Engagement: Utilizing Adapty's Placements for Paywalls and A/B Tests"

With Adapty's placement system, you can create and run paywalls and A/B tests at different points in your app user's journey, such as Onboarding, Settings, etc. These points are called Placements.

A placement in your app can manage multiple paywalls or A/B tests at a time, each made for a certain group of users, which we call Audiences. Moreover, you can experiment with paywalls, replacing one with another over time without releasing a new app version.

The only thing you hardcode in the mobile app is the placement ID.

To see the list of placements in Adapty, open the **Placements** section in the Adapty main menu.

The **Placements** list offers a comprehensive view of various locations in the user journey where paywalls or A/B tests can appear. Each item in the list corresponds to a specific placement, allowing easy management and modification. You can edit placement details, associate them with the desired paywall or A/B test for a specified audience, or remove unnecessary placements. The numbers in the table reflect the analytics for placements since their activation.

2023-07-26at_14.51.342x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

From here you can:

- Create a new placement
- Edit an existing placement
- Delete an existing placement
- Download local fallback paywalls, those are the paywalls that will be used when a user opens the app and there's no connection with Adapty backend (e.g. no internet connection or in the rare case when the backend is down) and there's no cache on the device.

---

title: "Choose meaningful placements" description: ""

metadataTitle: ""

When creating placements, it's essential to consider the logical flow of your app and the user experience you want to create. Most apps should have no more than 5 placements without sacrificing the ability to run experiments. Here's an example of how you can structure your placements:

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

1. **Onboarding placement:** This placement represents the first interaction your users have with your app. It's an excellent opportunity to introduce your users to your app's value proposition and set the stage for a positive user experience. Over 80% of subscriptions are activated during onboarding, so it's important to focus on selling the most profitable subscriptions here. With Adapty, you can easily show different paywalls to different audiences and run A/B tests to find the best option for your app. For example, you can run an A/B test for users from the US, showing more expensive subscriptions 50% of the time.
2. **App settings placement:** If the user hasn't subscribed during onboarding, you can create a placement within your app. This can be in the app settings or after the user has completed a specific target action. Since users inside the app tend to think more thoroughly about subscribing, the products on this paywall might be slightly less expensive compared to those in the onboarding stage.
3. **Promo placement:** If the user hasn't subscribed after seeing the paywall multiple times, it could indicate that the prices are too high for them or they are hesitant about subscriptions. In this case, you can show a special offer to them with the most affordable subscription or even a lifetime product. This can help entice users who are price-sensitive or skeptical about subscriptions to make a purchase.
   Most apps will have similar logic and placements, following the user journey and key points where paywalls or A/B tests can be displayed to drive conversions and revenue. In each placement, you can configure multiple paywalls or A/B tests to experiment and optimize your monetization strategies.

---

title: "Create placement" description: "Drive user interaction and optimize conversions by creating targeted placements for paywalls and A/B tests in your app with Adapty's user-friendly platform. Learn how to effortlessly deploy and manage placements for enhanced engagement"

metadataTitle: "Boost Engagement: Creating Placements in Adapty for Paywalls and A/B Tests"

A Placement designates a specific location within your mobile app where a paywall or A/B test can be displayed. For example, a subscription choice might appear in a startup flow, while a consumable product (such as golden coins) could be presented when a user runs out of coins in a game. You have the flexibility to showcase the same paywall or A/B test across multiple placements or different paywalls or A/B tests in one placement for different user segments, called audiences in Adapty.

Read the Choose meaningful placements section for more recommendations on how to choose the right placement for a paywall.

To create a new placement:

1. Open the **Placements** section in the Adapty main menu.

2. In the **Placements** window, click the **Create placement** button.

3. In the **Placements / New placement** window, enter **Placement Name ** that is used for your reference to clearly describe the exact place in your mobile app. This name serves as a means of identification and helps you organize and manage your placements effectively. You have the flexibility to edit the placement name even after it has been created.
4. Enter the **Placement ID** which is a unique identifier of the placement that will be used in your mobile app code to call the paywalls and A/B tests created in Adapty for this placement. Placement ID cannot be modified once it has been created. It guarantees the uniqueness and integrity of each placement.
5. Add a paywall or A/B test to the "All users" audience, which is automatically created for every placement.
6. Add more audiences and paywalls or A/B tests if required.
7. If you have more than one audience, check that the audiences have the correct priorities. When you have different user audiences, a user can belong to more than one audience. For instance, if you've defined audiences like "US users, "Facebook users", and a general audience like "All users," it's crucial to determine which specific audience to consider first when a user falls into multiple categories. Correct the priorities if required.
8. Click the **Save and publish button**.

---

title: "Edit placement" description: "Enhance user engagement by easily replacing paywalls or A/B tests and adjusting target user segments within placements. Streamline the process of updating displayed content without requiring app releases"

metadataTitle: "Efficient Paywall Replacement: Editing Placements in Adapty"

A Placement designates a specific location within your mobile app where a paywall or A/B test can be displayed. For example, a subscription choice might appear in a startup flow, while a consumable product (such as golden coins) could be presented when a user runs out of coins in a game. You have the flexibility to showcase the same paywall or A/B test across multiple placements or different paywalls or A/B tests in one placement for different user segments, called audiences in Adapty.

To edit an existing placement:

1. Open the **Placements** section in Adapty main menu.
2. In the **Placements** window, click the **3-dot** button next to the placement and select the **Edit** option.

3. In the opened **Edit** window, make the changes you need. For more details on the options in this window, please read the Create placement section.
4. Click the **Save and publish** button to confirm the changes.

---

title: "Delete placement" description: "Optimize app organization by effortlessly removing unused or mistakenly created placements with Adapty's deletion feature. Learn how to maintain a tidy environment and improve efficiency"

metadataTitle: "Efficient Placement Cleanup: Deleting Unused Placements in Adapty"

A Placement designates a specific location within your mobile app where a paywall or A/B test can be displayed.

:::danger Although you have the option to delete any placement, it is critical to ensure that you don't delete a placement that is actively used in your mobile app. Deleting such a placement will result in a local fallback paywall being permanently shown if you've set it up, and you won't be able to ever replace it with a dynamic paywall in released app versions. :::

To delete an existing placement:

1. Open the **Placements** section in Adapty main menu.
2. In the **Placements** window, click the **3-dot** button next to the placement and select the **Delete** option.

3. In the opened **Delete placement** window, enter the product name you're about to delete.

4. Click the **Delete forever** button to confirm the deletion.

---

title: "Audiences" description: "Enhance user engagement by creating personalized segments with Adapty's Audiences feature, enabling tailored paywalls or A/B tests for specific user groups based on various filters. Learn how to target users effectively and optimize content delivery"

metadataTitle: "Customized User Targeting: Using Audiences in Adapty"

Audiences in Adapty are user segments, helping you customize paywalls or A/B tests for specific groups of your users. You can set up these segments using special filters to make sure the right users see the right paywalls in your mobile app.

In Adapty, a **Placement** is where you can show paywalls or A/B tests. When you add an audience to a placement, you're targeting specific user groups with personalized content. For instance, you might show different paywalls based on a user's age, device, or subscription status. If a user falls into multiple groups, you can choose which group gets the priority, deciding which paywall they'll see.

In the example below, we have a placement with the identifier `Onboarding`. In your app code, you will access the placement using this identifier. If the user belongs to the "Yoga beginners" audience, they will see the first paywall Those who do not fit the "Yoga beginners" audience will see the second paywall.

To display a paywall or A/B test to a specific audience, do the following:

1. Create a user segment. You can skip this step if you want to show the paywall or A/B test to all users. In such a case, use the "All users" audience created by default.
2. Add this segment as an audience to placement and define which paywall or A/B test should be shown to it. The "All users" audience is automatically added to every placement; you only need to specify which paywall or A/B test should be displayed.
3. Set the right priorities if you have more than one audience in a placement. This ensures that users who belong to more than one audience will see the most relevant content. When a user is part of several audiences, the paywall for the highest-priority audience will be displayed.
4. In your mobile app code, show the paywall associated with this placement in the mobile app code.

---

title: "Add audience and paywall or A/B test to placement" description: "Optimize user engagement by linking audiences with paywalls or A/B tests within Adapty placements, enabling personalized content delivery for specific user groups based on various filters. Learn how to enhance targeted content delivery and improve user experience"

metadataTitle: "Targeted Content Delivery: Adding Audiences, paywalls, and A/B tests to Placements in Adapty"

Audiences in Adapty are segments of your customers that let you target the paywalls or A/B tests accurately. You can set up these segments using specific filters, making sure that the right paywalls show up for the right people.

When it comes to placement in Adapty, adding an audience to it means you're aiming specific contentâ€”like paywalls or A/B testsâ€”at certain user groups. Linking an audience with a placement ensures that the content you want is seen by the right users at the right points in their app journey.

1. Open the **Placements** section in the Adapty main menu.

2. In the **Placements** window, click the **Edit** button next to the placement where you wish to add an audience.

3. In the **Placements/ Your placement** window, click the **Add audience** button and choose the desired user segment from the list. Please note that only segments you have previously created are accessible in the list. These segments signify different audience groups defined and created within Adapty.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

4. Associate a paywall or A/B test to display for this audience. To do this, click either the **Run paywall** or **Run A/B test** button, then select the desired paywall or A/B test from the dropdown list. Please note that only previously created paywalls and previously created A/B tests are available in the lists.

   In Adapty, a paywall serves as a screen showcasing purchasable products within your mobile app, offering users the opportunity to make in-app purchases. To tailor the paywall experience, you can designate specific audiences to view particular paywalls. It's important to note that each audience is associated with only one paywall, ensuring a personalized approach. However, the same paywall can be presented to multiple audiences. For a broader reach, the default "All users" audience is available to display the paywall to everyone. This nuanced approach allows you to optimize your paywall strategies based on audience preferences and maximize user engagement.

   If you're uncertain about the effectiveness of the created paywall, consider comparing it with another one in an A/B test. Adapty provides the flexibility to enhance your monetization strategy further by introducing A/B tests. These tests involve presenting users with multiple paywalls to evaluate and determine the most effective one. Explore further insights on A/B tests in our A/B test documentation.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

5. Click the **Save and publish** button to confirm the addition of the audiences.

When multiple audiences are associated with the same placement, you can conveniently manage their priority.

---

title: "Change audience priority in placement" description: "Refine user engagement by adjusting audience priority levels within Adapty placements, guiding content selection for users belonging to multiple audiences. Learn how to optimize audience targeting and enhance content delivery efficiency"

metadataTitle: "Fine-Tune Audience Targeting: Changing Priority in Adapty Placements"

When you have different user audiences in one placement, a user can belong to more than one audience. For instance, if you've defined audiences like "Female", "Runners", and a general audience like "All users," it's crucial to determine which specific audience to consider first when a user falls into multiple categories.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

In this scenario, we rely on audience priority. Audience priority is a numerical order, where #1 is the highest. It guides the sequence for audiences to check. In simpler terms, audience priority helps Adapty make decisions about which audience to apply first when selecting the paywall or A/B test to display. If the audience priority for a paywall or A/B test is low, users who potentially qualify for the paywall or test might bypass it. Instead, they could be directed to another audience with a higher priority.

To adjust audience priorities for a placement:

1. Open the **Placements** section from the Adapty main menu.
2. Click the placement for which you want to change the audience priority.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

3. Click the **Edit placement** button.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

4. After the chosen placement opens with the list of its audiences, hover over any audience and click the **Edit** button once it shows.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

5. In the opened **Edit audience priorities** window, drag-and-drop audiences to reorder them correctly.
6. Click the **Save** button.

---

title: "Placement metrics" description: ""

metadataTitle: ""

With Adapty, you have the flexibility to create and manage multiple placements in your app, each associated with distinct paywalls or A/B tests. This versatility enables you to target specific user segments, experiment with different offers or pricing models, and optimize your app's monetization strategy.

To gather valuable insights into the performance of your placements and user engagement with your offers, Adapty tracks various user interactions and transactions related to the displayed paywalls. The robust analytics system captures metrics including views, unique views, purchases, trials, refunds, conversion rates, and revenue.

The collected metrics are continuously updated in real-time and can be conveniently accessed and analyzed through Adapty's user-friendly dashboard. You have the freedom to customize the time range for analysis, apply filters based on different parameters, and compare metrics across various placements, user segments, or products.

Placement metrics are available on the placements list, where you can get an overview of the performance of all your placements. This high-level view provides aggregated metrics for each placement, allowing you to compare their performance and identify trends.

For a more detailed analysis of each placements, you can navigate to the placements detail metrics. On this page, you will find comprehensive metrics specific to the selected placements. These metrics provide deeper insights into how a particular placements is performing, allowing you to assess its effectiveness and make data-driven decisions.

2023-07-26at_14.55.042x.png').default} style={{ border: 'none', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Metrics controls

The system displays the metrics based on the selected time period and organizes them according to the left-side column parameter with four levels of indentation.

View options for metrics data

The placement metrics page offers two view options for metrics data: paywall-based and audience-based.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

In the paywall-based view, metrics are grouped by placements associated with the paywall. This allows users to analyze metrics by different placements.

In the audience-based view, metrics are grouped by the target audience of the paywall. Users can assess metrics specific to different audience segments.

Profile install date filtration

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Time ranges

You can choose from a range of time periods to analyze metrics data, allowing you to focus on specific durations such as days, weeks, months, or custom date ranges.

2023-07-26at_16.49.272x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Available filters and grouping

Adapty offers powerful tools for filtering and customizing metrics analysis to suit your needs. With Adapty's metrics page, you have access to various time ranges, grouping options, and filtering possibilities.

- âœ… Filter by: Audience, paywall, paywall group, placement, country, store.
- âœ… Group by: Segment, store, and product

You can find more information about the available controls, filters, grouping options, and how to use them for paywall analytics in this documentation.

Single metrics chart

One of the key components of the placement metrics page is the chart section, which visually represents the selected metrics and facilitates easy analysis.

The chart section on the placements metrics page includes a horizontal bar chart that visually represents the chosen metric values. Each bar in the chart corresponds to a metric value and is proportional in size, making it easy to understand the data at a glance. The horizontal line indicates the timeframe being analyzed, and the vertical column displays the numeric values of the metrics. The total value of all the metric values is displayed next to the chart.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Additionally, clicking on the arrow icon in the top right corner of the chart section expands the view, displaying the selected metrics on the full line of the chart.

Total metrics summary

Next to the single metrics chart, the total metrics summary section is shown, which displays the cumulative values for the selected metrics at a specific point in time, with the ability for you to change the displayed metric using a dropdown menu.

2023-07-26at_14.55.492x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Metrics definitions

Unlock the power of placement metrics with our comprehensive definitions. From revenue to conversion rates, gain valuable insights that will supercharge your monetization strategies and drive success for your app.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Revenue

This metric represents the total amount of money generated in USD from purchases and renewals within specific placements. Please note that the revenue calculation does not include the Apple App Store or Google Play Store commission and is calculated before deducting any fees.

Proceeds

This metric represents the actual amount of money received by the app owner in USD from purchases and renewals within specific placements after deducting the applicable Apple App Store or Google Play Store commission. It reflects the net revenue that directly contributes to the app's earnings. For more information on how proceeds are calculated, you can refer to the Adapty documentation.

ARPPU

ARPPU stands for Average revenue per paying user and measures the average revenue generated per paying user within specific placements. It is calculated as the total revenue divided by the number of unique paying users. For example, if the total revenue is $15,000 and there are 1,000 paying users, the ARPPU would be $15.

ARPAS

ARPAS, or Average revenue per active subscriber, allows you to measure the average revenue generated per active subscriber within specific placements. It is calculated by dividing the total revenue by the number of subscribers who have activated a trial or subscription. For example, if the total revenue is $5,000 and there are 1,000 subscribers, the ARPAS would be $5. This metric helps assess the average monetization potential per subscriber.

Unique CR to purchases

The Unique conversion rate to purchases is calculated by dividing the number of purchases within specific placements by the number of unique views. It focuses on the ratio of purchases to the unique number of views, providing insights into the effectiveness of converting unique visitors within specific placements into paying customers.

CR to purchases

The Conversion rate to purchases is calculated by dividing the number of purchases within specific placements by the total number of views of paywalls. It indicates the percentage of views within specific placements that result in purchases, providing insights into the effectiveness of your paywall in converting users into paying customers.

Unique CR to trials

The unique conversion rate to trials is calculated by dividing the number of trials started within specific placements by the number of unique views. It measures the percentage of unique views within specific placements that result in trial activations, providing insights into the effectiveness of your paywall in converting unique visitors into trial users.

Purchases

Purchases represent the cumulative total of various transactions made on the paywall within specific placements. The following transactions are included in this metric (renewals are not included):

- New purchases are made directly within specific placements.
- Trial conversions of trials that were initially activated within specific placements.
- Downgrades, upgrades, and cross-grades of subscriptions made within specific placements.
- Subscription restores within specific placements, such as when a subscription is reinstated after expiration without auto-renewal.

By considering these different types of transactions, the purchases metric provides a comprehensive view of the overall acquisition and monetization activity within specific placements.

Trials

The trials metric represents the total number of trials that have been activated within specific placements. It reflects the number of users who have initiated trial periods through your paywall within those placements. This metric helps track the effectiveness of your trial offering and can provide insights into user engagement and conversion from trials to paid subscriptions.

Trials canceled

The trials canceled metric represents the number of trials within specific placements in which the auto-renewal feature has been switched off. This occurs when users manually unsubscribe from the trial, indicating their decision not to continue with the subscription after the trial period ends. Tracking trials canceled provides valuable information about user behavior and allows you to understand the rate at which users opt out of the trial within specific placements.

Refunds

The refunds metric represents the number of refunded purchases and subscriptions within specific placements. This includes transactions that have been reversed or refunded due to various reasons, such as customer requests, payment issues, or any other applicable refund policies.

Refund rate

The refund rate is calculated by dividing the number of refunds within specific placements by the number of first-time purchases (renewals are not included). For example, if there are 5 refunds and 1,000 first-time purchases, the refund rate would be 0.5%.

Views

The views metric represents the total number of times the paywall within specific placements has been viewed by users. Each time a user visits the paywall within those placements, it is counted as a separate view. Tracking views helps you understand the level of engagement and user interaction with your paywall, providing insights into user behavior and the effectiveness of your paywall placement and design within specific areas of your app.

Unique views

The unique views metric represents the number of unique instances in which the paywall within specific placements has been viewed by users. Unlike total views, which count each visit as a separate view, unique views count each user's visit to the paywall within those placements only once, regardless of how many times they access it. Tracking unique views helps provide a more accurate measure of user engagement and the reach of your paywall within specific placements, as it focuses on individual users rather than the total number of visits.

---

title: "Access levels" description: "Efficiently manage user access in your mobile app with Adapty's Access Levels feature, allowing flexible control over privileges based on product purchases. Explore how to create and customize access levels to tailor user experiences effectively"

metadataTitle: "Streamlined User Access Control: Managing Access Levels in Adapty"

Access levels let you control what your app's users can do in your mobile app without hardcoding specific product IDs. Each product defines how long the user gets a certain access level for. So, whenever a user makes a purchase, Adapty grants access to the app for a specific period (for subscriptions) or forever (for lifetime purchases).

When you create an app in the Adapty Dashboard, the `premium` access level is automatically generated. This serves as the default access level and it cannot be deleted.

You can have multiple access levels per app. This is when they can be useful:

- In a newspaper app where you sell subscriptions to different topics independently, you can create access levels such as `sports` and `science`.
- In a fitness app offering recorded video training under a regular subscription (using the default `premium` access level), customers may opt for a more expensive option providing access to live training with a coach. In this case, you can create a `live_coach_access` level.

- In a language learning app, you can choose to create an access level for each available language.

To begin working with access levels in Adapty, open the **Paywalls and Products** section from the Adapty main menu, then select the **Access levels** tab.

The **Access levels** list displays all access levels, including the `premium` one that is added automatically and those added by you in Adapty.

levellist.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Create access level" description: "Optimize user access management in your mobile app with Adapty's Access Levels feature, facilitating customization without reliance on specific product IDs. Learn how to create access levels for tailored user experiences"

metadataTitle: "Enhanced User Access Control: Creating Access Levels in Adapty"

Access levels let you control what your app's users can do in your mobile app without hardcoding specific product IDs. Each product defines how long the user gets a certain access level for. So, whenever a user makes a purchase, Adapty grants access to the app for a specific period (for subscriptions) or forever (for lifetime purchases).

When you create an app in the Adapty Dashboard, the `premium` access level is automatically generated. This serves as the default access level and it cannot be deleted.

To create a new access level:

1. Open the **Paywalls and Products** ***section from the Adapty main menu, then select the** **Access levels** tab.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **Create access level** button.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the **Create access level** window, assign it an ID. This ID will serve as the identifier within your mobile app, enabling access to additional features upon user purchase. Additionally, this identifier aids in distinguishing one access level from others within the app. Ensure it's clear and easily understandable for your convenience.
4. Click the **Create access level** button to confirm the access level creation.

---

title: "Assign access level to product" description: "Simplify user access control in your mobile app with Adapty's Access Levels feature, seamlessly assigning access levels to products for synchronized content access. Learn how to efficiently manage access levels and optimize user experiences"

metadataTitle: "Streamlined Access Management: Assigning Access Levels to Products in Adapty"

Every Product requires an associated access level to ensure that users receive the corresponding gated content upon purchase. Adapty seamlessly determines the subscription duration, which then serves as the expiration date for the access level. In the case of a lifetime product, if a customer purchases it, the access level remains perpetually active without any expiration date.

To link an access level to a product:

1. While configuring a product, select the access level from the **Access Level ID** list.

accesslevel*to*product.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **Save** button to confirm the change.

---

title: "Give access level to specific customer" description: "Enhance user satisfaction by manually assigning access levels to specific customers in Adapty, whether through the intuitive Adapty Dashboard interface or API integration. Learn how to personalize user experiences and optimize support interactions efficiently"

metadataTitle: "Personalized Access Control: Assigning Access Levels to Specific Customers in Adapty"

You can manually adjust the access level for a particular customer right in the Adapty Dashboard. This is useful, especially in support scenarios. For example, if you'd like to extend a user's premium usage by an extra week as a thank-you for leaving a fantastic review.

Give access level to a specific customer in the Adapty Dashboard

1. Open the **Profiles and Segments** section from the Adapty main menu, then select the **Profiles** tab.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. In the **Profiles** window, click on the customer you want to grant access to.

accesslevel*to*customer.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened window, click the **Add access level** button.

accesslevel*to*customer1.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. In the opened **Add Access level** window, select the Access level to grant and when it should expire for this customer.
5. Click the **Apply** button.

Give access level to a specific customer via API

You also have the option to give a customer an access level from your server using the Adapty API. This comes in handy if you have bonuses for referrals or other events related to your products. Find additional details in the API Specs section.

---

title: "Installation of Adapty SDKs" description: "Learn how to integrate AdaptySDK and AdaptyUI SDK into your mobile app for seamless functionality and effortless creation of subscription purchase pages with the Paywall builder. Get detailed installation and configuration guidance tailored for various frameworks"

metadataTitle: "AdaptySDK and AdaptyUI SDK Integration Guide for Mobile Apps"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Main **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

The Adapty SDK installation and configuration depend on your framework, so refer to the following doc topics for detailed guidance:

- Install and configure Adapty SDKs on iOS
- Install and configure Adapty SDKs on Android
- Install and configure Adapty SDKs on Flutter
- Install and configure Adapty SDKs on React Native
- Install and configure Adapty SDKs on Unity

---

title: "iOS - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on iOS, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "iOS - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual

constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI SDK version | | :---------- | :---------- | | 2.7.x, 2.8.x | 2.0.x | | 2.9.x - 2.10.0 | 2.1.2 | | 2.10.1 | 2.1.3 | | 2.10.3 and all later 2.10.x versions | 2.1.5 | | 2.11.1 | 2.11.1 | | 2.11.2 | 2.11.2 | | 2.11.3 | 2.11.3 | | 3.x | Included as a module within the Adapty SDK. For more information, see iOS - Adapty SDK v. 3.x installation & configuration. |

You can install AdaptySDK and AdaptyUI SDK via CocoaPods, or Swift Package Manager.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install SDKs via CocoaPods

1. Add Adapty to your `Podfile`:

   ```shell title="Podfile"
   pod 'Adapty', '~> 2.11.3'
   pod 'AdaptyUI', '~> 2.11.3'
   ```

2. Run:

   ```sh title="Shell"
   pod install
   ```

This creates a `.xcworkspace` file for your app. Use this file for all future development of your application.

Install SDKs via Swift Package Manager

1. In Xcode go to **File** -> **Add Package Dependency...**. Please note the way to add package dependencies can differ in XCode versions. Refer to XCode documentation if necessary.
2. Enter the repository URL `https://github.com/adaptyteam/AdaptySDK-iOS.git`
3. Choose the version, and click the **Add package** button. Xcode will add the package dependency to your project, and you can import it.
4. In the **Choose Package Products** window, click the **Add package** button once again. The package will appear in the **Packages** list.
5. Repeat steps 2-3 for AdaptyUI SDK URL: `https://github.com/adaptyteam/AdaptyUI-iOS.git`.

Configure Adapty SDK

You only need to configure the Adapty SDK once, typically early in your application lifecycle:

```swift // In your AppDelegate class:

let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) // optional .with(customerUserId: "YOUR*USER*ID") // optional .with(idfaCollectionDisabled: false) // optional .with(ipAddressCollectionDisabled: false) // optional

Adapty.activate(with: configurationBuilder) { error in // handle the error } </TabItem> <TabItem value="SwiftUI" label="SwiftUI" default> swift import Adapty

@main struct SampleApp: App { init() let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) // optional .with(customerUserId: "YOUR*USER*ID") // optional .with(idfaCollectionDisabled: false) // optional .with(ipAddressCollectionDisabled: false) // optional

    Adapty.activate(with: configurationBuilder) { error in
      // handle the error
    }
}

var body: some Scene {
    WindowGroup {
        ContentView()
    }
}

} ```
```

Parameters:

| Parameter | Presence | Description | |---------|--------|-----------| **PUBLIC*SDK*KEY** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection | | **observerMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics.

The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional | An identifier of the user in your system. We send it in subscription and analytical events, to attribute events to the right profile. You can also find customers by `customerUserId` in the **Profiles and Segments** menu. | | **idfaCollectionDisabled** | optional |

Set to `true` to disable IDFA collection and sharing.

the user IP address sharing.

The default value is `false`.

For more details on IDFA collection, refer to the Analytics integration section.

| | **ipAddressCollectionDisabled** | optional |

Set to `true` to disable user IP address collection and sharing.

The default value is `false`.

|

:::note - Note, that StoreKit 2 is available since iOS 15.0. Adapty will implement the legacy logic for older versions. - Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only. - **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one. :::

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` at any time in the application's lifespan, but we recommend that you do this before configuring Adapty.

```swift title="Swift"
Adapty.logLevel = .verbose
```

Redirect the logging system messages

If you for some reason need to send messages from Adapty to your system or save them to a file, you can override the default behavior:

```swift title="Swift"
Adapty.setLogHandler { level, message in writeToLocalFile("Adapty \(level): \(message)") }
```

---

title: "Android - Adapty SDK Installation and configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Android, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided"

metadataTitle: "Android - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI version | | :----------------- | :--------------- | | 2.7.xâ€“2.9.x | 2.0.x | | 2.10.0 | 2.1.2 | | 2.10.2 | 2.1.3 | | 2.11.x | 2.11.x |

You can install Adapty SDK via Gradle.

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install via Gradle

`groovy dependencies { ... implementation 'io.adapty:android-sdk:2.11.3' implementation 'io.adapty:android-ui:2.11.2' } kotlin dependencies { ... implementation("io.adapty:android-sdk:2.11.3") implementation("io.adapty:android-ui:2.11.1") }` ```toml //libs.versions.toml

[versions] .. adapty = "2.11.3" adaptyUi = "2.11.1"

[libraries] .. adapty = { group = "io.adapty", name = "android-sdk", version.ref = "adapty" } adapty-ui = { group = "io.adapty", name = "android-ui", version.ref = "adaptyUi" }

//module-level build.gradle.kts

dependencies { ... implementation(libs.adapty) implementation(libs.adapty.ui) } ```

If the dependency is not being resolved, please make sure that you have `mavenCentral()` in your Gradle scripts.

▶ The instruction on how to add it

If your project doesn't have `dependencyResolutionManagement` in your `settings.gradle`, add the following to your top-level `build.gradle` at the end of repositories:

`groovy title="top-level build.gradle" allprojects { repositories { ... mavenCentral() } }`

Otherwise, add the following to your `settings.gradle` in `repositories` of `dependencyResolutionManagement` section:

`groovy title="settings.gradle" dependencyResolutionManagement { ... repositories { ... mavenCentral() } }`

Configure Proguard

You should add `-keep class com.adapty.** { *; }` to your Proguard configuration.

Configure Adapty SDK

Add the following to your `Application` class:

```kotlin override fun onCreate() { super.onCreate() Adapty.activate( applicationContext, AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withObserverMode(false) //default false .withCustomerUserId(customerUserId) //default null .withIpAddressCollectionDisabled(false) //default false .build() )

```
//OR
//the method is deprecated since Adapty SDK v2.10.5

Adapty.activate(applicationContext, "PUBLIC_SDK_KEY", observerMode = false, customerUserId = "YOUR_USER_ID")
```

} ```

```java @Override public void onCreate() { super.onCreate(); Adapty.activate( applicationContext, new AdaptyConfig.Builder("PUBLIC*SDK*KEY") .withObserverMode(false) //default false .withCustomerUserId(customerUserId) //default null .withIpAddressCollectionDisabled(false) //default false .build() );

```
//OR
//the method is deprecated since Adapty SDK v2.10.5

Adapty.activate(getApplicationContext(), "PUBLIC_SDK_KEY", false, "YOUR_USER_ID");
```

} ```

Configurational options:

| Parameter | Presence | Description | |---------|--------|-----------| **PUBLIC*SDK*KEY** | required |

The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection.

Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only.

| | **observerMode** | optional |

A boolean value that controls Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional | An identifier of the user in your system. We send it in subscription and analytical events, to attribute events to the right profile. You can also find customers by `customerUserId` in the **Profiles and Segments** menu. If you don't have a user ID at the time of Adapty initialization, you can set it later using `.identify()` method. Read more in the Identifying users section. | | **IpAddressCollectionDisabled** | optional |

A boolean parameter. Set to `true` to disable the collection of the user IP address. The default value is `false`.

Parameter works with `AdaptyConfig.Builder` only.

|

:::note **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one. :::

Set up the logging system

Adapty logs errors and other important information to help you understand what is going on. There are the following levels available:

| Level | Description | | :---------------------- | :-- | | `AdaptyLogLevel.NONE` | Nothing will be logged. Default value | | `AdaptyLogLevel.ERROR` | Only errors will be logged | | `AdaptyLogLevel.WARN` | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | `AdaptyLogLevel.INFO` | Errors, warnings, and various information messages will be logged. | | `AdaptyLogLevel.VERBOSE` | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set the log level in your app before configuring Adapty.

`kotlin Adapty.logLevel = AdaptyLogLevel.VERBOSE java Adapty.setLogLevel(AdaptyLogLevel.VERBOSE);`

Redirect the logging system messages

If you for some reason need to send messages from Adapty to your system or save them to a file, you can override the default behavior:

`kotlin Adapty.setLogHandler { level, message -> //handle the log } java Adapty.setLogHandler((level, message) -> { //handle the log });`

title: "Flutter - Adapty SDK Installation and configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Flutter, enabling seamless integration of Adapty into your mobile app . Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "Flutter - Adapty SDK Installation and Configuration Guide"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI SDK version | | :---------- | :---------- | | 2.9.3 | 2.1.0 | | 2.10.0 | 2.1.1 | | 2.10.1 | 2.1.2 | | 2.10.2 | 2.1.3 |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install Adapty SDKs

1. Add Adapty and AdaptyUI to your `pubspec.yaml` file:

   ```
   yaml title="pubspec.yaml" dependencies: adapty_flutter: ^2.10.1 adapty_ui_flutter: ^2.1.1
   ```

2. Run:

   ```
   bash title="Bash" flutter pub get
   ```

3. Import Adapty SDKs in your application in the following way:

   ```
   dart title="Dart" import 'package:adapty_flutter/adapty_flutter.dart'; import 'package:adapty_ui_flutter/adapty_ui_flutter.dart';
   ```

Configure Adapty SDKs

The configuration of the Adapty SDK for Flutter slightly differs depending on the mobile operating system (iOS or Android) you are going to release it for.

Configure Adapty SDKs for iOS

Create `Adapty-Info.plist` and add it to your project. Add the flag `AdaptyPublicSdkKey` in this file with the value of your Public SDK key.

```
xml title="Adapty-Info.plist" <dict> <key>AdaptyPublicSdkKey</key> <string>PUBLIC_SDK_KEY</string> <key>AdaptyObserverMode</key> <false/> </dict>
```

Parameters:

| Parameter | Presence | Description | |---------|--------|----------| |**AdaptyPublicSdkKey** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings-> General** tab -> **API keys** subsection | | **AdaptyObserverMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. At any purchase or restore in your application, you'll need to call `.restorePurchases()` method to record the action in Adapty. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **idfaCollectionDisabled** | optional |

A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

Configure Adapty SDKs for Android

1. Add the `AdaptyPublicSdkKey` flag into the appâ€™s `AndroidManifest.xml` (Android) file with the value of your Public SDK key.

   ```
   xml title="AndroidManifest.xml" <application ...> ... <meta-data android:name="AdaptyPublicSdkKey" android:value="PUBLIC_SDK_KEY" /> <meta-data
   android:name="AdaptyObserverMode" android:value="false" /> </application>
   ```

   Required parameters:

| Parameter | Presence | Description | |---------|--------|----------| | PUBLIC*SDK*KEY | required |

Contents of the **Public SDK key** field in the **App Settings -> General** tab in the Adapty Dashboard. **SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one.

Make sure you use the **Public SDK key** for Adapty initialization, since the **Secret key** should be used for server-side API only.

| | AdaptyObserverMode | optional |

A boolean value that is controlling Observer mode . Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics.

The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | AdaptyIDFACollectionDisabled | optional |

A boolean parameter, that allows you to disable IDFA collection for your app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

2. In your application, add:

   ```
   javascript title="Flutter" import 'package:adapty_flutter/adapty_flutter.dart';
   ```

3. Activate Adapty SDK with the following code:

   ```
   javascript title="Flutter" try { Adapty().activate(); } on AdaptyError catch (adaptyError) {} } catch (e) {}
   ```

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` in your app before configuring Adapty.

```
javascript title="Flutter" try { await Adapty().setLogLevel(AdaptyLogLevel.verbose); } on AdaptyError catch (adaptyError) { } catch (e) {}
```

:::danger Read checklist before releasing the app

Before releasing your application, go through the Release Checklist to ensure that you have completed all the steps, and also check the success of the integration using the criteria for assessing its success. :::

---

title: "React Native - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on React Native, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

metadataTitle: "React Native - - Adapty SDK Installation and Configuration Guide"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

You currently need to have a `react-native-adapty` of version 2.4.7 or higher to use UI SDK.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| Adapty SDK version | AdaptyUI version | | :----------------- | :--------------- | | 2.7.0 â€“ 2.9.2 | 2.0.0 - 2.0.1 | | 2.9.3 - 2.9.8 | 2.1.0 | | 2.10.0 | 2.1.1 | | 2.10.1 | 2.1.2 | | 2.11.x | 2.11.0 |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

Install Adapty SDKs

Currently, React Native provides two development paths: Expo and Pure React Native. Adapty seamlessly integrates with both. Please refer to the section below that matches your chosen setup.

Install Adapty SDKs for Expo React Native

You can streamline your development process with Expo Application Services (EAS). While configuration may vary based on your setup, here you'll find the most common and straightforward setup available:

1. If you haven't installed the EAS Command-Line Interface (CLI) yet, you can do so by using the following command:

   ```
   sh title="Shell" npm install -g eas-cli
   ```

2. In the root of your project, install the dev client to make a development build:

   ```
   sh title="Shell" expo install expo-dev-client
   ```

3. Run the installation command:

   ```
   sh title="Shell" expo install react-native-adapty expo install @adapty/react-native-ui
   ```

4. For iOS: Make an iOS build with EAS CLI. This command may prompt you for additional info. You can refer to expo official documentation for more details:

   ```
   sh title="Shell" eas build --profile development --platform ios
   ```

5. For Android: Make an Android build with EAS CLI. This command may prompt you for additional info. You can refer to expo official documentation for more details:

   ```
   sh title="Shell" eas build --profile development --platform android
   ```

6. Start a development server with the following command:

   ```
   sh title="Shell" expo start --dev-client
   ```

This should result in the working app with react-native-adapty.

Possible errors:

| Error | Description | |-----|----------| |`Failed to start (Invariant Violation: Native module cannot be null)` |

if you scan a QR code from a CLI dev client it might lead you to this error. To resolve it you can try the following:

> On your device open EAS built app (it should provide some Expo screen) and manually insert the URL that Expo provides (screenshot below). You can unescape special characters in URL with the JS function `unescape(â€œstringâ€)`, which should result in something like `http://192.168.1.35:8081`

|

Install Adapty SDKs with Pure React Native

If you opt for a purely native approach, please consult the following instructions:

1. In your project, run the installation command:

   ```
   sh title="Shell" yarn add react-native-adapty yarn add @adapty/react-native-ui
   ```

2. For iOS: Install required pods:

   ```
   sh title="Shell" pod install --project-directory=ios pod install --project-directory=ios/
   ```

   The minimal supported iOS version is 13.0. If you encounter an error during pod installation, locate this line in your`ios/Podfile` and update the minimal target. Tand update the minimum target. Afterward, you should be able to successfully execute `pod install`.

   ```
   diff title="Podfile" -platform :ios, min_ios_version_supported +platform :ios, 13.0
   ```

3. For Android: Update the `/android/build.gradle` file. Make sure there is the `kotlin-gradle-plugin:1.8.0` dependency or a newer one:

   ```
   groovy title="/android/build.gradle" ... buildscript { ... dependencies { ... classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.8.0" } } ...
   ```

Configure Adapty SDKs

To use Adapty SDKs, import `adapty` and invoke `activate` in your *core component* such as `App.tsx`. Preferably, position the activation before the React component to ensure no other Adapty calls occur before the activation.

```typescript title="/src/App.tsx" import { adapty } from 'react-native-adapty';

adapty.activate('PUBLIC*SDK*KEY');

const App = () => { // ... } ```

You can pass several optional parameters during activation:

```
typescript adapty.activate('PUBLIC_SDK_KEY', { observerMode: false, customerUserId: 'YOUR_USER_ID', logLevel: 'error', __debugDeferActivation: false, ipAddressCollectionDisabled: false, ios: { idfaCollectionDisabled: false, }, }); ```javascript import { IosStorekit2Usage, LogLevel } from 'react-native-adapty';

adapty.activate('PUBLIC*SDK*KEY', { observerMode: false, customerUserId: 'YOUR*USER*ID', logLevel: LogLevel.ERROR, __debugDeferActivation: false, ipAddressCollectionDisabled: false, ios: { idfaCollectionDisabled: false, }, }); ```
```

Activation parameters:

| Parameter | Presence | Description | |---------|--------|----------| **PUBLIC*SDK*KEY** | required |

A Public SDK Key is the unique identifier used to integrate Adapty into your mobile app. You can copy it in the Adapty Dashboard: **App settings -> General *tab -> *API Keys** section.

**SDK keys** are unique for every app, so if you have multiple apps make sure you choose the right one.
```

Make sure you use the **Public SDK key** for the Adapty initialization, since the **Secret key** should be used for the server-side API only.

| | **observerMode** | optional |

A boolean value controlling Observer mode . Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸš§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **customerUserId** | optional |

An identifier of a user in your system. We send it with subscription and analytical events, so we can match events to the right user profile. You can also find customers using the `customerUserId` in the Profiles section.

If you don't have a user ID when you start with Adapty, you can add it later using the `adapty.identify()` method. For more details, see the Identifying users section.

| | **logLevel** | optional | A string parameter that makes Adapty record errors and other important information to help you understand what's happening. | | **__debugDeferActivation** | optional | A boolean parameter, that lets you delay SDK activation until your next Adapty call. This is intended solely for development purposes and **should not be used in production**. | | **ipAddressCollectionDisabled** | optional |

Set to `true` to disable user IP address collection and sharing.

The default value is `false`.

| | **idfaCollectionDisabled** | optional | A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`. For more details, refer to the Analytics integration section. |

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other crucial information to provide insight into your app's functionality. There are the following available levels:

| Level | Description | | :------ | :---------- | | error | Only errors will be logged. | | warn | Errors and messages from the SDK that do not cause critical errors, but are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can set `logLevel` at any time in the application's lifespan, but we recommend that you do this before configuring Adapty.

`typescript adapty.setLogLevel('verbose');` ```javascript import { LogLevel } from 'react-native-adapty';

adapty.setLogLevel(LogLevel.VERBOSE); ```

For both `activate` and `setLogLevel` methods TypeScript validates the string you pass as an argument. However, if you're using JavaScript, you may prefer to use the `LogLevel` enum, that would guarantee to provide you a safe value:

Handle logs

If you're storing your standard output logs, you might wish to distinguish Adapty logs from others. You can achieve this by appending a prefix to all `AdaptyError` instances that are logged:

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.prefix = "[ADAPTY]"; ```

You can also handle all raised errors from any location you prefer using `onError`. Errors will be thrown where expected, but they will also be duplicated to your event listener.

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.onError = error => { // ... console.error(error); }; ```

## Delay SDK activation for development purposes

Adapty pre-fetches all necessary user data upon SDK activation, enabling faster access to fresh data.

However, this may pose a problem in the iOS simulator, which frequently prompts for authentication during development. Although Adapty cannot control the StoreKit authentication flow, it can defer the requests made by the SDK to obtain fresh user data.

By enabling the `__debugDeferActivation` property, the activate call is held until you make the next Adapty SDK call. This prevents unnecessary prompts for authentication data if not needed.

It's important to note that **this feature is intended for development use only**, as it does not cover all potential user scenarios. In production, activation should not be delayed, as real devices typically remember authentication data and do not repeatedly prompt for credentials.

Here's the recommended approach for usage:

`typescript title="Typescript" adapty.activate('PUBLIC_SDK_KEY', { __debugDeferActivation: isSimulator(), // 'isSimulator' from any 3rd party library });`

---

title: "Unity - Adapty SDK installation & configuration" description: "Discover step-by-step instructions for installing and configuring Adapty SDK and AdaptyUI SDK on Unity, enabling seamless integration of Adapty into your mobile app. Find the correct pair of SDKs with the compatibility table provided."

## metadataTitle: "Unity - - Adapty SDK Installation and Configuration Guide"

Adapty comprises two crucial SDKs for seamless integration into your mobile app:

- Core **AdaptySDK**: This is a fundamental, mandatory SDK necessary for the proper functioning of Adapty within your app.
- **AdaptyUI SDK**: This optional SDK becomes necessary if you use the Adapty Paywall builder: a user-friendly, no-code tool for easily creating cross-platform paywalls. These paywalls are built in a visual constructor right in our dashboard, run entirely natively on the device, and require minimal effort from you to create something that performs well.

Please consult the compatibility table below to choose the correct pair of Adapty SDK and AdaptyUI SDK.

| AdaptySDK-Unity version | AdaptyUI-Unity version | | :---------------------- | :---------------------- | | 2.7.1 | 2.0.1 | | 2.9.0 | not compatible |

:::danger Go through release checklist before releasing your app

Before releasing your application, make sure to carefully review the Release Checklist thoroughly. This checklist ensures that you've completed all necessary steps and provides criteria for evaluating the success of your integration. :::

## Install Adapty SDKs

To install the Adapty SDKs:

1. Download the `adapty-unity-plugin-*.unitypackage` from GitHub and import it into your project.

./ width: '400px', / image width */display: 'block',* / for alignment */margin: '0 auto'* / center alignment */ }} />

2. Download the `adapty-ui-unity-plugin-*.unitypackage` from GitHub and import it into your project.

./ width: '400px', / image width */display: 'block',* / for alignment */margin: '0 auto'* / center alignment */ }} />

3. Download and import the External Dependency Manager plugin.
4. The SDK uses the "External Dependency Manager" plugin to handle iOS Cocoapods dependencies and Android gradle dependencies. After the installation, you may need to invoke the dependency manager:

   ```
   Assets -> External Dependency Manager -> Android Resolver -> Force Resolve
   ```

   and

   ```
   Assets -> External Dependency Manager -> iOS Resolver -> Install Cocoapods
   ```

5. When building your Unity project for iOS, you would get `Unity-iPhone.xcworkspace` file, which you have to open instead of `Unity-iPhone.xcodeproj`, otherwise, Cocoapods dependencies won't be used.

Configure Adapty SDKs

To configure the Adapty SDKs for Unity, start by initializing the Adapty Unity Plugin and then using it as described in the guidance below. Additionally, ensure to set up your logging system to receive errors and other important information from Adapty.

Initiate Adapty Unity Plugin on iOS

The Adapty Unity Plugin on iOS is initialized automatically. To make it work properly:

1. Manually create the `Adapty-Info.plist` file and add it to the `/Assets` folder of your Unity project. It will be automatically copied to the Xcode project during the build phase. Below is an example of how this file should be structured:

```
xml title="Xml" <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>AdaptyPublicSdkKey</key> <string>insert_here_your_Adapty_public_key#</string> <key>AdaptyObserverMode</key> <false/> </dict> </plist>
```

Parameters:

| Parameter | Presence | Description | |---------|--------|-----------| |**AdaptyPublicSdkKey** | required | The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings**-> **General** tab -> **API keys** subsection | | **AdaptyObserverMode** | optional |

A boolean value controlling Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. The default value is `false`.

ðŸ§§ When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

| | **idfaCollectionDisabled** | optional |

A boolean parameter, that allows you to disable IDFA collection for your iOS app. The default value is `false`.

For more details, refer to the Analytics integration section.

|

Initiate Adapty Unity Plugin on Android

The Adapty Unity Plugin on Android is automatically initialized. To ensure it works properly:

1. Add `<meta-data` section with "AdaptyPublicSdkKey" as a direct child of the `<application` section to your project's AndroidManifest.xml file. If you don't have one, it can be easily created in**Project Settings** -> **Player** -> **Settings for Android** -> **Publishing settings** -> **Custom Main Manifest** checkbox). Here is an example:

```
```xml title="Xml"

    <meta-data
        android:name="AdaptyPublicSdkKey"
        android:value="PUBLIC_SDK_KEY" />

     <meta-data
         android:name="AdaptyObserverMode"
         android:value="false" />
</application>

```
```

Configurational options:

| Parameter | Presence | Description | |---------|--------|-----------| |**AdaptyPublicSDKkey** | required |

The key you can find in the **Public SDK key** field of your app settings in Adapty: **App settings**-> **General** tab -> **API keys** subsection.

Make sure you use the **Public SDK key** for Adapty initialization, the **Secret key** should be used for server-side API only.

| | **AdaptyObservermode** | optional |

A boolean value that controls Observer mode. Turn it on if you handle purchases and subscription status yourself and use Adapty for sending subscription events and analytics. Default value is `false`.

Note: When running in Observer mode, Adapty SDK won't close any transactions, so make sure you're handling it.

|

Use Adapty Unity Plugin

1. Create a script to listen to Adapty events. Name it `AdaptyListener` in your scene. We suggest using the `DontDestroyOnLoad` method for this object to ensure it persists throughout the application's lifespan.

adaptylistener.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty uses `AdaptySDK` namespace. At the top of your script files that use the Adapty SDK, you may add

```
csharp title="C#" using AdaptySDK;
```

2. Subscribe to Adapty events:

```csharp title="C#" using UnityEngine; using AdaptySDK;

public class AdaptyListener : MonoBehaviour, AdaptyEventListener { void Start() { DontDestroyOnLoad(this.gameObject); Adapty.SetEventListener(this); }

 public void OnLoadLatestProfile(Adapty.Profile profile) {
   // handle updated profile data
 }

} ```
```

Please keep in mind that for paywalls and products to be displayed in your mobile application, and for analytics to work, you need to display the paywalls and, if you're using paywalls not created with the Paywall Builder, handle the purchase process within your app.

Set up the logging system

Adapty logs errors and other important information to help you understand what is going on. There are three levels available:

| error | Only errors will be logged. | | :------ | :---------- | | warn | Errors and messages from the SDK that do not cause critical errors are worth paying attention to will be logged. | | info | Errors, warnings, and serious information messages, such as those that log the lifecycle of various modules will be logged. | | verbose | Any additional information that may be useful during debugging, such as function calls, API queries, etc. will be logged. |

You can call `SetLogLevel()` method in your app before configuring Adapty.

---

title: "Display paywalls designed with Paywall Builder" description: ""

metadataTitle: ""

import Details from '@site/src/components/Details';

With Adapty, you have the flexibility to configure paywalls remotely, defining the products to display in your app. This streamlined process eliminates the need to hardcode your products.

There are 2 ways to customize a paywall in the Adapty Dashboard:

- simple no-code tool called **Paywall Builder**
- flexible **remote config**, a JSON with data needed to render the paywall on the device

This topic describes the flow only for **Paywall Builder paywalls**. Displaying and handling the interactive parts of the paywall is different for remote config paywalls, so if you want to know more about it, please refer to the Display remote config paywalls topic.

If you've designed a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains what should be shown within the paywall, how it should be shown, and how to process user's actions like making purchases. Moreover, this paywall takes care of logging the paywall view event on its own, so you don't need to worry about ensuring analytics receives the data.

However, some coding still remains necessary. For instance, you'll need to retrieve the paywall itself, display it within the mobile app, and respond to the events it generates.

▶ Before you start displaying paywalls (click to expand)

1. Create your products in the Adapty dashboard
2. Create a paywall in the Adapty Dashboard and incorporate the products into it
3. Create a placement and incorporate your paywall into it
4. Install AdaptySDK and AdaptyUI SDK.

How to display and process paywalls created in the Paywall Builder

1. Fetch the paywall to show in the specific placement.
2. Show the paywall.
3. Handle the events produced by the paywall.

---

title: "Fetch Paywall Builder paywalls and their configuration" description: "Learn how to fetch paywalls and products for remote config paywalls in your app, crucial for displaying the right content to users based on their placements."

metadataTitle: "Learn how to fetch paywalls and products for remote config paywalls in your app"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

After you designed the visual part for your paywall with Paywall Builder in the Adapty Dashboard, you can display it in your mobile app. The first step in this process is to get the paywall associated with the placement and its view configuration as described below.

Please be aware that this topic refers to Paywall Builder-customized paywalls. For guidance on fetching remote config paywalls, please refer to the Fetch paywalls and products for remote config paywalls in your mobile app topic.

▶ Before you start displaying paywalls in your mobile app (click to expand)

1. Create your products in the Adapty Dashboard.
2. Create a paywall and incorporate the products into it in the Adapty Dashboard.
3. Create placements and incorporate your paywall into it in the Adapty Dashboard.
4. Install Adapty SDK and AdaptyUI DSK in your mobile app.

Fetch paywall designed with Paywall Builder

If you've designed a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown. Nevertheless, you need to get its ID via the placement, its view configuration, and then present it in your mobile app.

To ensure optimal performance, it's crucial to retrieve the paywall and its view configuration as early as possible, allowing sufficient time for images to download before presenting them to the user.

To get a paywall, use the `getPaywall` method:

`swift Adapty.getPaywall(placementId: "YOUR_PLACEMENT_ID", locale: "en") { result in switch result { case let .success(paywall): // the requested paywall case let .failure(error): // handle the error } }` ```kotlin import com.adapty.utils.seconds

...

Adapty.getPaywall("YOUR*PLACEMENT*ID", locale = "en", loadTimeout = 10.seconds) { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall } is AdaptyResult.Error -> { val error = result.error // handle the error } } } `</TabItem> <TabItem value="java" label="Java" default>` java import com.adapty.utils.TimeInterval;

...

Adapty.getPaywall("YOUR*PLACEMENT*ID", "en", TimeInterval.seconds(10), result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue(); // the requested paywall

```
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}
```

}); `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript try { final paywall = await Adapty().getPaywall(id: "YOUR*PLACEMENT*ID", locale: "en"); // the requested paywall } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp Adapty.GetPaywall("YOUR*PLACEMENT*ID", "en", (paywall, error) => { if(error != null) { // handle the error return; }

// paywall - the resulting object }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript try { const id = 'YOUR*PLACEMENT*ID'; const locale = 'en';

`const paywall = await adapty.getPaywall(id, locale);`

// the requested paywall } catch (error) { // handle the error } ```

| Parameter | Presence | Description | |---------|--------|-----------| **placementId** | required | The identifier of the desired Placement. This is the value you specified when creating a placement in the Adapty Dashboard. | | **locale** |

optional

default: `en`

|

The identifier of the paywall localization. This parameter is expected to be a language code composed of one or two subtags separated by the minus (-) character. The first subtag is for the language, the second one is for the region.

Example: `en` means English, `pt-br` represents the Brazilian Portuguese language.

See Localizations and locale codes for more information on locale codes and how we recommend using them.

| | **fetchPolicy** | default: `.reloadRevalidatingCacheData` |

By default, SDK will try to load data from the server and will return cached data in case of failure. We recommend this variant because it ensures your users always get the most up-to-date data.

However, if you believe your users deal with unstable internet, consider using `.returnCacheDataElseLoad` to return cached data if it exists. In this scenario, users might not get the absolute latest data, but they'll experience faster loading times, no matter how patchy their internet connection is. The cache is updated regularly, so it's safe to use it during the session to avoid network requests.

Note that the cache remains intact upon restarting the app and is only cleared when the app is reinstalled or through manual cleanup.

Adapty SDK stores paywalls locally in two layers: regularly updated cache described above and fallback paywalls. We also use CDN to fetch paywalls faster and a stand-alone fallback server in case the CDN is unreachable. This system is designed to make sure you always get the latest version of your paywalls while ensuring reliability even in cases where internet connection is scarce.

| | **loadTimeout** | default: 5 sec |

This value limits the timeout for this method. If the timeout is reached, cached data or local fallback will be returned.

Note that in rare cases this method can timeout slightly later than specified in `loadTimeout`, since the operation may consist of different requests under the hood.

For Android: You can create `TimeInterval` with extension functions (like `5.seconds`, where `.seconds` is from `import com.adapty.utils.seconds`), or `TimeInterval.seconds(5)`. To set no limitation, use `TimeInterval.INFINITE`.

|

Don't hardcode product IDs! Since paywalls are configured remotely, the available products, the number of products, and special offers (such as free trials) can change over time. Make sure your code handles these scenarios.
For example, if you initially retrieve 2 products, your app should display those 2 products. However, if you later retrieve 3 products, your app should display all 3 without requiring any code changes. The only thing you should hardcode is the placement ID.

Response parameters:

| Parameter | Description | | :-------- | :---------- | | Paywall | An `AdaptyPaywall` object with a list of prod IDs, the paywall identifier, remote config, and several other properties. |

Fetch the view configuration of paywall designed using Paywall Builder

After fetching the paywall, check if it includes a `ViewConfiguration`, which indicates that it was created using Paywall Builder. This will guide you on how to display the paywall. If the `ViewConfiguration` is present, treat it as a Paywall Builder paywall; if not, handle it as a remote config paywall.

For paywalls with a `ViewConfiguration`, use the `getViewConfiguration` method to load the view configuration. In cross-platform SDKs, you can directly call the `createPaywallView` method without manually fetching the view configuration first.

:::warning The result of the `createPaywallView` method can be used only once. If you need to reuse it, call the `createPaywallView` method again. :::

```swift import Adapty

guard paywall.hasViewConfiguration else { // use your custom logic return }

AdaptyUI.getViewConfiguration(forPaywall: paywall) { result in switch result { case let .success(viewConfiguration): // use loaded configuration case let .failure(error): // handle the error } } </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin if (!paywall.hasViewConfiguration) { // use your custom logic return }

AdaptyUI.getViewConfiguration(paywall) { result -> when(result) { is AdaptyResult.Success -> { val viewConfiguration = result.value // use loaded configuration } is AdaptyResult.Error -> { val error = result.error // handle the error } } } </TabItem> <TabItem value="java" label="Java" default> java if (!paywall.hasViewConfiguration()) { // use your custom logic return; }

AdaptyUI.getViewConfiguration(paywall, result -> { if (result instanceof AdaptyResult.Success) { AdaptyUI.ViewConfiguration viewConfiguration = ((AdaptyResult.Success) result).getValue(); // use loaded configuration } else if (result instanceof AdaptyResult.Error) { AdaptyError error = ((AdaptyResult.Error) result).getError(); // handle the error } }); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript import 'package:adapty*ui*flutter/adapty*ui*flutter.dart';

try { final view = await AdaptyUI().createPaywallView(paywall: paywall); } on AdaptyError catch (e) { // handle the error } catch (e) { // handle the error } </TabItem> <TabItem value="Unity" label="Unity" default> typescript import {createPaywallView} from '@adapty/react-native-ui';

if (paywall.hasViewConfiguration) { try { const view = await createPaywallView(paywall); } catch (error) { // handle the error } } else { //use your custom logic } </TabItem> <TabItem value="RN" label="React Native (TS)" default> csharp AdaptyUI.CreatePaywallView(paywall, preloadProducts: true, (view, error) => { // use the view }); ```

:::note If you are using multiple languages, learn how to add a Paywall builder localization and how to use locale codes correctly here. :::

Once you have successfully loaded the paywall and its view configuration, you can proceed to presenting the paywall in your mobile app.

## Speed up paywall fetching with default audience paywall

Typically, paywalls are fetched almost instantly, so you donâ€™t need to worry about speeding up this process. However, in cases where you have numerous audiences and paywalls, and your users have a weak internet connection, fetching a paywall may take longer than you'd like. In such situations, you might want to display a default paywall to ensure a smooth user experience rather than showing no paywall at all.

To address this, you can use the `getPaywallForDefaultAudience` method, which fetches the paywall of the specified placement for the **All Users** audience. However, it's crucial to understand that the recommended approach is to fetch the paywall by the `getPaywall` method, as detailed in the Fetch Paywall Information section above.

:::warning Why we recommend using `getPaywall`

The `getPaywallForDefaultAudience` method comes with a few significant drawbacks:

- **Potential backward compatibility issues**: If you need to show different paywalls for different app versions (current and future), you may face challenges. Youâ€™ll either have to design paywalls that support the current (legacy) version or accept that users with the current (legacy) version might encounter issues with non-rendered paywalls.
- **Loss of targeting**: All users will see the same paywall designed for the **All Users** audience, which means you lose personalized targeting (including based on countries, marketing attribution or your own custom attributes).

If you're willing to accept these drawbacks to benefit from faster paywall fetching, use the `getPaywallForDefaultAudience` method as follows. Otherwise stick to `getPaywall` described above. :::

```
swift Adapty.getPaywallForDefaultAudience(placementId: "YOUR_PLACEMENT_ID", locale: "en") { result in switch result { case let .success(paywall): // the requested paywall case let .failure(error): // handle the error } } kotlin Adapty.getPaywallForDefaultAudience("YOUR_PLACEMENT_ID", locale = "en") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java
Adapty.getPaywallForDefaultAudience("YOUR*PLACEMENT*ID", "en", result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue(); // the requested paywall

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> typescript try { const id = 'YOUR*PLACEMENT*ID'; const locale = 'en';

const paywall = await adapty.getPaywallForDefaultAudience(id, locale);

// the requested paywall } catch (error) { // handle the error } ```

:::note The `getPaywallForDefaultAudience` method is available starting from these versions:

- iOS: 2.11.2
- Android: 2.11.3
- React Native: 2.11.2

The method is not yet supported in Flutter and Unity, but support will be added soon. :::

| Parameter | Presence | Description | |---------|--------|-----------| **placementId** | required | The identifier of the Placement. This is the value you specified when creating a placement in your Adapty Dashboard. | | **locale** |

optional

default: `en`

|

The identifier of the paywall localization. This parameter is expected to be a language code composed of one or more subtags separated by the minus (**-**) character. The first subtag is for the language, the second one is for the region.

Example: `en` means English, `pt-br` represents the Brazilian Portuguese language.

See Localizations and locale codes for more information on locale codes and how we recommend using them.

| | **fetchPolicy** | default: `.reloadRevalidatingCacheData` |

By default, SDK will try to load data from the server and will return cached data in case of failure. We recommend this variant because it ensures your users always get the most up-to-date data.

However, if you believe your users deal with unstable internet, consider using `.returnCacheDataElseLoad` to return cached data if it exists. In this scenario, users might not get the absolute latest data, but they'll experience faster loading times, no matter how patchy their internet connection is. The cache is updated regularly, so it's safe to use it during the session to avoid network requests.

Note that the cache remains intact upon restarting the app and is only cleared when the app is reinstalled or through manual cleanup.

|

---

title: "Present Paywall Builder paywalls" description: ""

metadataTitle: ""

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

For detailed guidance on presenting paywalls within different frameworks, please refer to the specified topics on each framework:

- iOS - Present paywalls
- Android - Present paywalls
- Flutter - Present paywalls
- React Native - Present paywalls
- Unity - Present paywalls

---

title: "iOS - Present Paywall Builder paywalls" description: ""

metadataTitle: ""

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

:::note If you work in Observer mode, refer to the iOS - Present Paywall Builder paywalls in Observer mode topic instead. :::

Present paywalls in Swift

In order to display the visual paywall on the device screen, you must first configure it. To do this, use the method `.paywallController(for:products:viewConfiguration:delegate:)`:

```swift title="Swift" import Adapty import AdaptyUI

let visualPaywall = AdaptyUI.paywallController( for: , products: , viewConfiguration: , delegate: ) ```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Paywall** | required | An `AdaptyPaywall` object to obtain a controller for the desired paywall. | | **Products** | optional | Provide an arr of `AdaptyPaywallProducts` to optimize the display timing of products on the screen. If `nil` is passed, AdaptyUI will automatically fetch the necessary products. | | **ViewConfiguration** | required | An `AdaptyUI.LocalizedViewConfiguration` object containing visual details of the paywall. Use the `AdaptyUI.getViewConfiguration(paywall:locale:)` method. Refer to Fetch Paywall Builder paywalls and their configuration topic for more details. | | **Delegate** | required | An `AdaptyPaywallControllerDelegate` to listen to paywall events. Refer to Handling paywall events topic for more details. | | **TagResolver** | optional | Define a dictionary of custom tags and their resolved values. Custom tags serve as placeholders in the paywall content, dynamically replaced with specific strings for personalized content within the paywall. Refer to Custom tags in paywall builder topic for more details. |

Returns:

| Object | Description | | :---------- | :---------- | | AdaptyPaywallController | An object, representing the requested paywall screen |

After the object has been successfully created, you can display it on the screen of the device:

```
swift title="Swift" present(visualPaywall, animated: true)
```

## Present paywalls in SwiftUI

In order to display the visual paywall on the device screen, use the `.paywall` modifier in SwiftUI:

```swift title="SwiftUI" @State var paywallPresented = false

var body: some View { Text("Hello, AdaptyUI!") .paywall( isPresented: $paywallPresented, paywall: , configuration: , didPerformAction: { action in switch action { case .close: paywallPresented = false default: // Handle other actions break } }, didFinishPurchase: { product, profile in paywallPresented = false }, didFailPurchase: { product, error in /* handle the error */, didFinishRestore: { profile in */* check access level and dismiss */}, didFailRestore: { error in */* handle the error */ }, didFailRendering: { error in paywallPresented = false } ) } ```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Paywall** | required | An `AdaptyPaywall` object to obtain a controller for the desired paywall. | | **Product** | optional | Provide an array `AdaptyPaywallProducts` to optimize the display timing of products on the screen. If `nil` is passed, AdaptyUI will automatically fetch the necessary products. | | **Configuration** | required | An `AdaptyUI.LocalizedViewConfiguration` object containing visual details of the paywall. Use the `AdaptyUI.getViewConfiguration(paywall:locale:)` method. Refer to Fetch Paywall Builder paywalls and their configuration topic for more details. | | **TagResolver** | optional | Define a dictionary of custom tags and their resolved values. Custom tags serve as placeholders in the paywall content, dynamically replaced with specific strings for personalized content within the paywall. Refer to Custom tags in paywall builder topic for more details. |

Closure parameters:

| Closure parameter | Description | | :---------- | :---------- | | **didFinishPurchase** | If Adapty.makePurchase() succeeds, this callback will be invoked. | | **didFailPurchase** | If Adapty.makePurchase() fails, this callback will be invoked. | | **didFinishRestore** | If Adapty.restorePurchases() succeeds, this callback will be invoked. | | **didFailRestore** | If Adapty.restorePurchases() fails, this callback will be invoked. | | **didFailRendering** | If an error occurs during the interface rendering, this callback will be invoked. |

Refer to the iOS - Handling events topic for other closure parameters.

---

title: "Android - Present Paywall Builder paywalls" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

:::note If you work in Observer mode, refer to the Android - Present Paywall Builder paywalls in Observer mode topic instead. :::

In order to display the visual paywall on the device screen, you must first configure it. To do this, call the method `AdaptyUI.getPaywallView()` or create the `AdaptyPaywallView` directly:

```kotlin val paywallView = AdaptyUI.getPaywallView( activity, viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), eventListener, personalizedOfferResolver, tagResolver, )

//======= OR =======

val paywallView = AdaptyPaywallView(activity) // or retrieve it from xml ... with(paywallView) { setEventListener(eventListener) setObserverModeHandler(observerModeHandler) showPaywall( viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), personalizedOfferResolver, tagResolver, ) } ```

```java AdaptyPaywallView paywallView = AdaptyUI.getPaywallView( activity, viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), eventListener, personalizedOfferResolver, tagResolver );

//======= OR =======

AdaptyPaywallView paywallView = new AdaptyPaywallView(activity); //add to the view hierarchy if needed, or you receive it from xml ... paywallView.setEventListener(eventListener); paywallView.showPaywall(viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), personalizedOfferResolver); `</TabItem> <TabItem value="XML" label="XML" default>` xml width="matchparent" android:layout*height="match*parent" /> ```

After the view has been successfully created, you can add it to the view hierarchy and display it on the screen of the device.

If you get `AdaptyPaywallView` *not* by calling `AdaptyUI.getPaywallView()`, you will also need to call `.setEventListener()` and `.showPaywall()` methods.

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Paywall** | required | Specify an `AdaptyPaywall` object, for which you are trying screen representation. | | **Products** | optional | Provide an array of `AdaptyPaywallProduct`to optimize the display timing of products on the screen. If `null` is passed, AdaptyUI will automatically fetch the required products. | | **ViewConfiguration** | required | Supply an `AdaptyViewConfiguration` object containing visual details of the paywall. Use the `Adapty.getViewConfiguration(paywall)` method to load it. Refer to Fetch the visual configuration of paywall topic for more details. | | **Insets** | required | Define an `AdaptyPaywallInsets` object containing information about the area overlapped by system bars, creating vertical margins for content. If neither the status bar nor the navigation bar overlaps the `AdaptyPaywallView`, pass `AdaptyPaywallInsets.NONE`. For fullscreen mode where system bars overlap part of your UI, obtain insets as shown under the table. | | **EventListener** | optional | Provide an `AdaptyUiEventListener` to observe paywall events. Extending AdaptyUiDefaultEventListener is recommended for ease of use. Refer to Handling paywall events topic for more details. | | **PersonalizedOfferResolver** | optional | To indicate personalized pricing (read more ), implement `AdaptyUiPersonalizedOfferResolver` and pass your own logic that maps `AdaptyPaywallProduct` to true if the

product's price is personalized, otherwise false. | | **TagResolver** | optional | Use `AdaptyUiTagResolver` to resolve custom tags within the paywall text. This resolver takes a tag parameter and resolves it to a corresponding string. Refer to Custom tags in paywall builder topic for more details. |

For fullscreen mode where system bars overlap part of your UI, obtain insets in the following way:

```kotlin import androidx.core.graphics.Insets import androidx.core.view.ViewCompat import androidx.core.view.WindowInsetsCompat

//create extension function fun View.onReceiveSystemBarsInsets(action: (insets: Insets) -> Unit) { ViewCompat.setOnApplyWindowInsetsListener(this) { _, insets -> val systemBarInsets = insets.getInsets(WindowInsetsCompat.Type.systemBars()) ViewCompat.setOnApplyWindowInsetsListener(this, null) action(systemBarInsets) insets } } //and then use it with the view paywallView.onReceiveSystemBarsInsets { insets -> val paywallInsets = AdaptyPaywallInsets.of(insets.top, insets.bottom) paywallView.showPaywall(paywall, products, viewConfig, paywallInsets, productTitleResolver) } </TabItem> <TabItem value="java" label="Java" default> java import androidx.core.graphics.Insets; import androidx.core.view.ViewCompat; import androidx.core.view.WindowInsetsCompat;

...

ViewCompat.setOnApplyWindowInsetsListener(paywallView, (view, insets) -> { Insets systemBarInsets = insets.getInsets(WindowInsetsCompat.Type.systemBars()); ViewCompat.setOnApplyWindowInsetsListener(paywallView, null);

AdaptyPaywallInsets paywallInsets = AdaptyPaywallInsets.of(systemBarInsets.top, systemBarInsets.bottom); paywallView.showPaywall(paywall, products, viewConfiguration, paywallInsets, productTitleResolver);

return insets;

}); ```

Returns:

| Object | Description | | :---------- | :---------- | | `AdaptyPaywallView` | object, representing the requested paywall screen. |

---

title: "Flutter - Present Paywall Builder paywalls" description: ""

## metadataTitle: ""

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

To show a paywall, call `view.present()` method. You can use `view` from the previous step, we will introduce a new one for visibility reasons:

typescript title="Flutter" try { await view.present(); } on AdaptyError catch (e) { // handle the error } catch (e) { // handle the error }

---

title: "React Native - Present Paywall Builder paywalls" description: ""

## metadataTitle: ""

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

To display a paywall view, simply call the `view.present()` method. If you've already defined a `view` object in a previous step, feel free to use it. In the following snippet, we'll introduce a new `view` for better visibility.

```typescript title="React Native (TSX)" import {createPaywallView} from '@adapty/react-native-ui';

const view = await createPaywallView(paywall);

view.registerEventHandlers(); // handle close press, etc

try { await view.present(); } catch (error) { // handle the error }

```

---

title: "Unity - Present Paywall Builder paywalls" description: ""

metadataTitle: ""

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

To display a paywall view, simply call the `view.present()` method. If you've already defined a `view` object in a previous step, feel free to use it. In the following snippet, we'll introduce a new `view` for better visibility.

csharp title="Unity" view.Present((error) => { // handle the error });

---

title: "Hide Paywall Builder paywalls (on cross-platform SDKs)" description: ""

metadataTitle: ""

While Paywall Builder seamlessly handles the purchasing process upon clicking "buy" buttons, you have to manage the closure of paywall screens within your mobile app.

In native iOS and Android SDKs you have complete control over both presenting and hiding the paywalls. However in Flutter, React Native, and Unity SDKs this works a bit differently. Learn how below.

Dismiss a paywall screen in Flutter

You can hide a paywall screen by calling the `view.dismiss` method.

typescript title="Flutter" try { await view.dismiss(); } on AdaptyError catch (e) { // handle the error } catch (e) { // handle the error }

Dismiss a paywall screen in React Native

You can hide a paywall view in 2 ways:

- call the `view.dismiss` method
- return `true` from any event ve-handlinhandler.

```typescript title="React Native (TSX)" try { await view.dismiss(); } catch (error) { // handle the error }

```

Dismiss a paywall screen in Unity

You can hide a paywall view by calling the `view.Dismiss` method.

typescript title="Flutter" view.Dismiss((error) => { // handle the error });

---

title: "Handle paywall events" description: ""

metadataTitle: ""

Paywalls designed with the Paywall Builder generate some events that require your attention and handling after they are raised. Among such events are paywall closure, URL opening, product selection, purchase start, purchase cancellation, successful and failed purchase, as well as successful and failed purchase restoration.

In the framework-specific sections, we'll dive into details of effective management and monitoring processes taking place on the paywall screen within your mobile app using the Adapty SDK. Explore the configuration guides for every platform below:
```

- iOS - Handling events
- Android - Handling events
- Flutter - Handling events
- React Native - Handling events
- Unity - Handling events

---

title: "iOS - Handle paywall events" description: ""

metadataTitle: ""

Paywalls configured with the Paywall Builder don't need extra code to make and restore purchases. However, they generate some events that your app can respond to. Those events include button presses (close buttons, URLs, product selections, and so on) as well as notifications on purchase-related actions taken on the paywall. Learn how to respond to these events below.

Handling events in Swift

To control or monitor processes occurring on the paywall screen within your mobile app, implement the `AdaptyPaywallControllerDelegate` methods.

User-generated events

Actions

If a user performs some action (`close`, `openURL(url:)` or `custom(id:)`), the method below will be invoked. Note that this is just an example and you can implement the response to actions differently:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didPerform action: AdaptyUI.Action) {

```
switch action {
    case .close:
        controller.dismiss(animated: true)
    case let .openURL(url):
            // handle URL opens (incl. terms and privacy links)
        UIApplication.shared.open(url, options: [:])
    case let .custom(id):
        if id == "login" {
            // implement login flow
        }
        break
}
```

} ```

For example, if a user taps the close button, the action `close` will occur and you are supposed to dismiss the paywall. Note that at the very least you need to implement the reactions to both `close` and `openURL`.

> ðŸ’¡ Login Action
>
> If you have configured Login Action in the dashboard, you should also implement reaction for custom action with id `"login"`.

#### Product selection

If a user selects a product for purchase, this method will be invoked:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didSelectProduct product: AdaptyPaywallProduct) { }
```

Started purchase

If a user initiates the purchase process, this method will be invoked:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didStartPurchase product: AdaptyPaywallProduct) { }
```

It will not be invoked in Observer mode. Refer to the iOS - Present Paywall Builder paywalls in Observer mode topic for details.

Canceled purchase

If a user initiates the purchase process but manually interrupts it, the method below will be invoked. This event occurs when the `Adapty.makePurchase()` function completes with a `.paymentCancelled` error:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didCancelPurchase product: AdaptyPaywallProduct) { }
```

It will not be invoked in Observer mode. Refer to the iOS - Present Paywall Builder paywalls in Observer mode topic for details.

Successful purchase

If `Adapty.makePurchase()` succeeds, this method will be invoked:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didFinishPurchase product: AdaptyPaywallProduct, purchasedInfo: AdaptyPurchasedInfo) {
controller.dismiss(animated: true) }
```

We recommend dismissing the paywall screen in that case.

It will not be invoked in Observer mode. Refer to the iOS - Present Paywall Builder paywalls in Observer mode topic for details.

Failed purchase

If `Adapty.makePurchase()` fails, this method will be invoked:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didFailPurchase product: AdaptyPaywallProduct, error: AdaptyError) { }
```

It will not be invoked in Observer mode. Refer to the iOS - Present Paywall Builder paywalls in Observer mode topic for details.

Successful restore

If `Adapty.restorePurchases()` succeeds, this method will be invoked:

```swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didFinishRestoreWith profile: AdaptyProfile) { }
```

We recommend dismissing the screen if a the has the required `accessLevel`. Refer to the Subscription status topic to learn how to check it.

Failed restore

If `Adapty.restorePurchases()` fails, this method will be invoked:

```swift title="Swift" public func paywallController(_ controller: AdaptyPaywallController, didFailRestoreWith error: AdaptyError) { }
```

Data fetching and rendering

Product loading errors

If you don't pass the product array during the initialization, AdaptyUI will retrieve the necessary objects from the server by itself. If this operation fails, AdaptyUI will report the error by calling this method:

```swift title="Swift" public func paywallController(_ controller: AdaptyPaywallController, didFailLoadingProductsWith error: AdaptyError) -> Bool { return true }
```

If you return `true`, AdaptyUI will repeat the request after 2 seconds.

Rendering errors

If an error occurs during the interface rendering, it will be reported by this method:

```swift title="Swift" public func paywallController(_ controller: AdaptyPaywallController, didFailRenderingWith error: AdaptyError) { }
```
```

In a normal situation, such errors should not occur, so if you come across one, please let us know.

Handling events in SwiftUI

To control or monitor processes occurring on the paywall screen within your mobile app, use the `.paywall` modifier in SwiftUI:

```swift title="Swift" @State var paywallPresented = false

var body: some View { Text("Hello, AdaptyUI!") .paywall( isPresented: $paywallPresented, paywall: paywall, configuration: viewConfig, didPerformAction: { action in switch action { case .close: paywallPresented = false case .openURL(url): // handle opening the URL (incl. for terms and privacy) case default: // handle other actions break } }, didSelectProduct: { /* Handle the event */ }, *didStartPurchase: { /* Handle the event */ }, didFinishPurchase: { product, info in /* Handle the event */ }, didFailPurchase: { product, error in /* Handle the event */ }, didCancelPurchase: { /* Handle the event */ }, didStartRestore: { /* Handle the event */ }, didFinishRestore: { /* Handle the event */ }, didFailRestore: { /* Handle the event */* }, didFailRendering: { error in paywallPresented = false }, didFailLoadingProducts: { error in return false } ) } ```

You can register only the closure parameters you need, and omit those you do not need. In this case, unused closure parameters will not be created.

| Closure parameter | Description | | :---------- | :---------- | | **didSelectProduct** | If a user selects a product for purchase, this parameter will be invoked. | | **didStartPurchase** | If a user initiates the purchase process, this parameter will be invoked. | | **didFinishPurchase** | If Adapty.makePurchase() succeeds, this parameter will be invoked. | | **didFailPurchase** | If Adapty.makePurchase() fails, this parameter will be invoked. | | **didCancelPurchase** | If a user initiates the purchase process but manually interrupts it, this parameter will be invoked. | | **didStartRestore** | If a user initiates the purchase restoration, this parameter will be invoked. | | **didFinishRestore** | If `Adapty.restorePurchases()` succeeds, this parameter will be invoked. | | **didFailRestore** | If `Adapty.restorePurchases()` fails, this parameter will be invoked. | | **didFailRendering** | If an error occurs during the interface rendering, this parameter will be invoked. | | **didFailLoadingProducts** | If you don't pass the product array during the initialization, AdaptyUI will retrieve the necessary objects from the server by itself. If this operation fails, AdaptyUI will invoke this parameter. |

Note that at the very least you need to implement the reactions to both `close` and `openURL`.

---

title: "Android - Handle paywall events" description: ""

## metadataTitle: ""

Paywalls configured with the Paywall Builder don't need extra code to make and restore purchases. However, they generate some events that your app can respond to. Those events include button presses (close buttons, URLs, product selections, and so on) as well as notifications on purchase-related actions taken on the paywall. Learn how to respond to these events below.

If you need to control or monitor the processes that take place on the purchase screen, implement the `AdaptyUiEventListener` methods.

If you would like to leave the default behavior in some cases, you can extend `AdaptyUiDefaultEventListener` and override only those methods you want to change.

Below are the defaults from `AdaptyUiDefaultEventListener`.

### User-generated events

#### Actions

If a user has performed some action (`Close`, `OpenURL` or `Custom`, this method will be invoked:

```kotlin title="Kotlin" override fun onActionPerformed(action: AdaptyUI.Action, view: AdaptyPaywallView) { when (action) { AdaptyUI.Action.Close -> (view.context as? Activity)?.onBackPressed()

    is AdaptyUI.Action.OpenUrl -> //launching intent to open url

    is AdaptyUI.Action.Custom -> //no default action
}

} ```

The following action types are supported:

- `Close`
- `OpenUrl(url)`
- `Custom(id)`

Note that at the very least you need to implement the reactions to both `Close` and `OpenURL`. For example, if a user taps the close button, the action `Close` will occur and you are supposed to dismiss the paywall.

:::warning This method is *not* invoked when user taps the system back button instead of the close icon on the screen. :::

> ðŸ’¡ Login Action
>
> If you have configured Login Action in the dashboard, you should implement reaction for custom action with id `"login"`

#### Product selection

If a product is selected for purchase (by a user or by the system), this method will be invoked:

```kotlin title="Kotlin" public override fun onProductSelected( product: AdaptyPaywallProduct, view: AdaptyPaywallView, ) {}```

#### Started purchase

If a user initiates the purchase process, this method will be invoked:

```kotlin title="Kotlin" public override fun onPurchaseStarted( product: AdaptyPaywallProduct, view: AdaptyPaywallView, ) {}```

The method will not be invoked in Observer mode. Refer to the Android - Present Paywall Builder paywalls in Observer mode topic for details.

#### Canceled purchase

If a user initiates the purchase process but manually interrupts it afterward, the method below will be invoked. This event occurs when the `Adapty.makePurchase()` function completes with the `USER_CANCELED` error:

```kotlin title="Kotlin" public override fun onPurchaseCanceled( product: AdaptyPaywallProduct, view: AdaptyPaywallView, ) {}```

The method will not be invoked in Observer mode. Refer to the Android - Present Paywall Builder paywalls in Observer mode topic for details.

#### Successful purchase

If `Adapty.makePurchase()` succeeds, this method will be invoked:

```kotlin title="Kotlin" public override fun onPurchaseSuccess( profile: AdaptyProfile?, product: AdaptyPaywallProduct, view: AdaptyPaywallView, ) { (view.context as? Activity)?.onBackPressed() }```

We recommend dismissing the screen in that case.

The method will not be invoked in Observer mode. Refer to the Android - Present Paywall Builder paywalls in Observer mode topic for details.

#### Failed purchase

If `Adapty.makePurchase()` fails, this method will be invoked:

```kotlin title="Kotlin" public override fun onPurchaseFailure( error: AdaptyError, product: AdaptyPaywallProduct, view: AdaptyPaywallView, ) {}```

The method will not be invoked in Observer mode. Refer to the Android - Present Paywall Builder paywalls in Observer mode topic for details.

#### Successful restore

If `Adapty.restorePurchases()` succeeds, this method will be invoked:

```kotlin title="Kotlin" public override fun onRestoreSuccess( profile: AdaptyProfile, view: AdaptyPaywallView, ) {}```

We recommend dismissing the screen if the user has the required `accessLevel`. Refer to the Subscription status topic to learn how to check it.

#### Failed restore

If `Adapty.restorePurchases()` fails, this method will be invoked:

```
kotlin title="Kotlin" public override fun onRestoreFailure(error: AdaptyError, view: AdaptyPaywallView,) {}
```

Data fetching and rendering

Product loading errors

If you don't pass the products during the initialization, AdaptyUI will retrieve the necessary objects from the server by itself. If this operation fails, AdaptyUI will report the error by invoking this method:

```
kotlin title="Kotlin" public override fun onLoadingProductsFailure( error: AdaptyError, view: AdaptyPaywallView, ): Boolean = false
```

If you return `true`, AdaptyUI will repeat the request in 2 seconds.

Rendering errors

If an error occurs during the interface rendering, it will be reported by calling this method:

```
kotlin title="Kotlin" public override fun onRenderingError( error: AdaptyError, view: AdaptyPaywallView, ) {}
```

In a normal situation, such errors should not occur, so if you come across one, please let us know.

---

title: "Flutter - Handle paywall events" description: ""

metadataTitle: ""

Paywalls configured with the Paywall Builder don't need extra code to make and restore purchases. However, they generate some events that your app can respond to. Those events include button presses (close buttons, URLs, product selections, and so on) as well as notifications on purchase-related actions taken on the paywall. Learn how to respond to these events below.

To control or monitor processes occurring on the paywall screen within your mobile app, implement the `AdaptyUIObserver` methods and register the observer before presenting any screen:

```
javascript title="Flutter" AdaptyUI().addObserver(this);
```

User-generated events

Actions

If a user has performed some action (`close`, `openURL`, `androidSystemBack`, or `custom`, this method will be invoked:

```javascript title="Flutter" // You have to install url*launcher plugin in order to handle urls: // https://pub.dev/packages/url*launcher import 'package:url*launcher/url*launcher_string.dart';

void paywallViewDidPerformAction(AdaptyUIView view, AdaptyUIAction action) { switch (action.type) { case AdaptyUIActionType.close: view.dismiss(); break; case AdaptyUIActionType.openUrl: final urlString = action.value; if (urlString != null) { launchUrlString(urlString); } default: break; } } ```

The following action types are supported:

- `close`
- `openUrl`
- `custom`
- `androidSystemBack`.

Note that at the very least you need to implement the reactions to both `close` and `openURL`.

For example, if a user taps the close button, the action `close` will occur and you are supposed to dismiss the paywall. Refer to the Hide Paywall Builder paywalls topic for details on dismissing a paywall screen. Note that `AdaptyUIAction` has optional value property: look at this in the case of `openUrl` and `custom`.

> ðŸ’¡ Login Action
>
> If you have configured Login Action in the dashboard, you should implement reaction for `custom` action with value `"login"`

#### Product selection

If a product is selected for purchase (by a user or by the system), this method will be invoked:

```
javascript title="Flutter" void paywallViewDidSelectProduct(AdaptyUIView view, AdaptyPaywallProduct product) { }
```

#### Started purchase

If a user initiates the purchase process, this method will be invoked:

```
javascript title="Flutter" void paywallViewDidStartPurchase(AdaptyUIView view, AdaptyPaywallProduct product) { }
```

Canceled purchase

If a user initiates the purchase process but manually interrupts it, the function below will be invoked. Basically, this event occurs when the `Adapty.makePurchase()` function completes with the `.paymentCancelled` error:

```
javascript title="Flutter" void paywallViewDidCancelPurchase(AdaptyUIView view, AdaptyPaywallProduct product) { }
```

Successful purchase

If `Adapty.makePurchase()` succeeds, this method will be invoked:

```
javascript title="Flutter" void paywallViewDidFinishPurchase(AdaptyUIView view, AdaptyPaywallProduct product, AdaptyProfile profile) { }
```

We recommend dismissing the screen in that case. Refer to the Hide Paywall Builder paywalls for details on dismissing a paywall screen.

Failed purchase

If `Adapty.makePurchase()` fails, this method will be invoked:

```
javascript title="Flutter" void paywallViewDidFailPurchase(AdaptyUIView view, AdaptyPaywallProduct product, AdaptyError error) { }
```

Successful restore

If `Adapty.restorePurchases()` succeeds, this method will be invoked:

```
javascript title="Flutter" void paywallViewDidFinishRestore(AdaptyUIView view, AdaptyProfile profile) { }
```

We recommend dismissing the screen if the user has the required `accessLevel`. Refer to the Subscription status topic to learn how to check it and to Hide Paywall Builder paywalls topic to learn how to dismiss a paywall screen.

Failed restore

If `Adapty.restorePurchases()` fails, this method will be invoked:

```
javascript title="Flutter" void paywallViewDidFailRestore(AdaptyUIView view, AdaptyError error) { }
```

Data fetching and rendering

Product loading errors

If you don't pass the product array during the initialization, AdaptyUI will retrieve the necessary objects from the server by itself. If this operation fails, AdaptyUI will report the error by invoking this method:

```
javascript title="Flutter" void paywallViewDidFailLoadingProducts(AdaptyUIView view, AdaptyIOSProductsFetchPolicy? fetchPolicy, AdaptyError error) { }
```
```

Rendering errors

If an error occurs during the interface rendering, it will be reported by calling this method:

```
javascript title="Flutter" void paywallViewDidFailRendering(AdaptyUIView view, AdaptyError error) { }
```

In a normal situation, such errors should not occur, so if you come across one, please let us know.

---

title: "React Native - Handle paywall events" description: ""

metadataTitle: ""

Paywalls configured with the Paywall Builder don't need extra code to make and restore purchases. However, they generate some events that your app can respond to. Those events include button presses (close buttons, URLs, product selections, and so on) as well as notifications on purchase-related actions taken on the paywall. Learn how to respond to these events below.

To control or monitor processes occurring on the paywall screen within your mobile app, implement the`view.registerEventHandlers` method:

```typescript title="React Native (TSX)" import {createPaywallView} from '@adapty/react-native-ui';

const view = await createPaywallView(paywall);

const unsubscribe = view.registerEventHandlers({ onCloseButtonPress() { return true; }, onPurchaseCompleted(profile) { return true; }, onPurchaseStarted(product) { *//}, onPurchaseCancelled(product) {//* }, onPurchaseFailed(error) { *//}, onRestoreCompleted(profile) {//* }, onRestoreFailed(error) { *//}, onProductSelected() {//*}, onRenderingFailed(error) { *//}, onLoadingProductsFailed(error) {//* }, onUrlPress(url) { /* handle url */ }, });

```

You can register event handlers you need, and miss those you do not need. In this case, unused event listeners would not be created.

Note that at the very least you need to implement the reactions to both `onCloseButtonPress` and `onUrlPress`.

Event handlers return a boolean. If `true` is returned, the displaying process is considered complete, thus the paywall screen closes and event listeners for this view are removed.

Note, that `onCloseButtonPress`, `onPurchaseCompleted` and `onRestoreCompleted` in the example above return `true` â€" This is their default behavior that you can override.

Event handlers

| Event handler | Description | | :---------- | :---------- | | **onCloseButtonPress** | If the close button is visible and a user taps it, this method will be invoked. It is recommended to dismiss the paywall screen in this handler. | | **onPurchaseCompleted** | If a user's purchase succeeds, this method will be invoked and will provide updated `AdaptyProfile`. It is recommended to dismiss the paywall view in this handler. | | **onPurchaseStarted** | If a user taps the "Purchase" action button to start the purchase process, this method will be invoked and will provide `AdaptyPaywallProduct`. | | **onPurchaseCancelled** | If a user initiates the purchase process and manually interrupts it, this method will be invoked and will provide `AdaptyPaywallProduct`. | | **onPurchaseFailed** | If the purchase process fails, this method will be invoked and provide `AdaptyError`. | | **onRestoreCompleted** | If a user's purchase restoration succeeds, this method will be invoked and provide updated `AdaptyProfile`. It is recommended to dismiss the screen if the user has the required `accessLevel`. Refer to the Subscription status topic to learn how to check it. | | **onRestoreFailed** | If the restoring process fails, this method will be invoked and will provide `AdaptyError`. | | **onProductSelected** | When any product in the paywall view is selected, this method will be invoked, so that you can monitor what the user selects before the purchase. | | **onRenderingFailed** | If an error occurs during view rendering, this method will be invoked and provide `AdaptyError`. Such errors should not occur, so if you come across one, please let us know. | | **onLoadingProductsFailed** | If you haven't set `prefetchProducts: true` in view creation, AdaptyUI will retrieve the necessary objects from the server by itself. If this operation fails, this method will be invoked and provide `AdaptyError`. |

---

title: "Unity - Handle paywall events" description: ""

metadataTitle: ""

Paywalls configured with the Paywall Builder don't need extra code to make and restore purchases. However, they generate some events that your app can respond to. Those events include button presses (close buttons, URLs, product selections, and so on) as well as notifications on purchase-related actions taken on the paywall. Learn how to respond to these events below.

To control or monitor processes occurring on the paywall screen within your mobile app, implement the `AdaptyUIEventListener` methods and register the observer before presenting any screen:

```
csharp title="Unity" AdaptyUI.SetEventListener(this);
```

User-generated events

Actions

If a user has performed some action, this method will be invoked:

```
csharp title="Unity" public void OnPerformAction(AdaptyUI.View view, AdaptyUI.Action action) { switch (action.Type) { case AdaptyUI.ActionType.Close: view.Dismiss(null); break; case AdaptyUI.ActionType.OpenUrl: var urlString = action.Value; if (urlString != null { Application.OpenURL(urlString); } default: // handle other events break; } }
```

The following action types are supported:

- `Close`
- `OpenUrl`
- `Custom`

- `AndroidSystemBack`.

  At the very least you need to implement the reactions to both `close` and `openURL`.

For example, if a user taps the close button, the action `Close` will occur and you are supposed to dismiss the paywall.
Note that `AdaptyUI.Action` has optional value property: look at this in the case of `OpenUrl` and `Custom`.

  ðŸ'¡ Login Action

  If you have configured Login Action in the dashboard, you should implement reaction for `Custom` action with value `"login"`

Product selection

If a product was selected for purchase (by a user or by the system), this method will be invoked.

```
csharp title="Unity" public void OnSelectProduct(AdaptyUI.View view, Adapty.PaywallProduct product) { }
```

Started purchase

If a user initiates the purchase process, this method will be invoked.

```
csharp title="Unity" public void OnStartPurchase(AdaptyUI.View view, Adapty.PaywallProduct product) { }
```

Canceled purchase

If a user initiates the purchase process but manually interrupts it, this method will be invoked. This event occurs when the `Adapty.MakePurchase()` function completes with a `.paymentCancelled` error:

```
csharp title="Unity" public void OnCancelPurchase(AdaptyUI.View view, Adapty.PaywallProduct product) { }
```

Successful purchase

If `Adapty.MakePurchase()` succeeds, this method will be invoked:

```
csharp title="Unity" public void OnFinishPurchase(AdaptyUI.View view, Adapty.PaywallProduct product, Adapty.Profile profile) { }
```

We recommend dismissing the screen in that case.

Failed purchase

If `Adapty.MakePurchase()` fails, this method will be invoked:

```
csharp title="Unity" public void OnFailPurchase(AdaptyUI.View view, Adapty.PaywallProduct product, Adapty.Error error) { }
```

Successful restore

If `Adapty.RestorePurchases()` succeeds, this method will be invoked:

```
csharp title="Unity" public void OnFinishRestore(AdaptyUI.View view, Adapty.Profile profile) { }
```

We recommend dismissing the screen if the user has the required `accessLevel`. Refer to the Subscription status topic to learn how to check it.

Failed restore

If `Adapty.RestorePurchases()` fails, this method will be invoked:

```
csharp title="Unity" public void OnFailRestore(AdaptyUI.View view, Adapty.Error error) { }
```

Data fetching and rendering

Product loading errors

If you didn't pass the product array during initialization, AdaptyUI will retrieve the necessary objects from the server by itself. In this case, this operation may fail, and AdaptyUI will report the error by invoking this method:

```
csharp title="Unity" public void OnFailLoadingProducts(AdaptyUI.View view, Adapty.Error error) { }
```

Rendering errors

If an error occurs during the interface rendering, it will be reported by calling this method:

```
csharp title="Unity" public void OnFailRendering(AdaptyUI.View view, Adapty.Error error) { }
```

In a normal situation, such errors should not occur, so if you come across one, please let us know.

---

title: "Display paywalls designed with remote config" description: ""

metadataTitle: ""

import Details from '@site/src/components/Details';

With Adapty, you have the flexibility to configure paywalls remotely, defining the products your app will display. This process eliminates the need to hardcode your products. The only thing you hardcode is the placement ID.

There are 2 ways to customize a paywall in the Adapty Dashboard:

- a simple no-code tool called **Paywall Builder**
- flexible **remote config**, a JSON with data needed to render the paywall on the device

This topic describes the flow only for **Remote Config paywalls**. Displaying and handling the interactive parts of the paywall is different for Paywall Builder paywalls, so if you want to know more about it, please consult the Display paywalls designed with Paywall Builder topic.

▶ Before you start displaying paywalls (click to expand)

1. Create your products in the Adapty dashboard.
2. Create a paywall in the Adapty Dashboard and incorporate the products into your paywall.
3. Create placements and incorporate your paywall into the placement.
4. Install Adapty SDK in your mobile app.

How to display and process paywalls designed by the remote config

1. Fetch the paywall and products to show in the specific placement.
2. Render the paywall.

---

title: "Fetch paywalls and products for remote config paywalls" description: "Learn how to fetch paywalls and products for remote config paywalls in your app, crucial for displaying the right content to users based on their placements."

metadataTitle: "Learn how to fetch paywalls and products for remote config paywalls in your app"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

Before showcasing remote config and custom paywalls, you need to fetch the information about them. Please be aware that this topic refers to remote config and custom paywalls. For guidance on fetching paywalls for Paywall Builder-customized paywalls, please consult Fetch Paywall Builder paywalls and their configuration.

▶ Before you start fetching paywalls and products in your mobile app (click to expand)

1. Create your products in the Adapty Dashboard.
2. Create a paywall and incorporate the products into your paywall in the Adapty Dashboard.
3. Create placements and incorporate your paywall into the placement in the Adapty Dashboard.
4. Install Adapty SDK in your mobile app.

Fetch paywall information

In Adapty, a product serves as a combination of products from both the App Store and Google Play. These cross-platform products are integrated into paywalls, enabling you to showcase them within specific mobile app placements.

To display the products, you need to obtain a Paywall from one of your placements with `getPaywall` method.

```
swift Adapty.getPaywall(placementId: "YOUR_PLACEMENT_ID", locale: "en") { result in switch result { case let .success(paywall): // the requested paywall case let .failure(error): // handle the error } } kotlin Adapty.getPaywall("YOUR_PLACEMENT_ID", locale = "en") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getPaywall("YOURPLACEMENTID", "en", result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue(); // the requested paywall

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final paywall = await Adapty().getPaywall(id: "YOURPLACEMENTID", locale: "en"); // the requested paywall } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.GetPaywall("YOURPLACEMENTID", "en", (paywall, error) => { if(error != null) { // handle the error return; }

// paywall - the resulting object }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const id = 'YOURPLACEMENTID'; const locale = 'en';

const paywall = await adapty.getPaywall(id, locale);

// the requested paywall } catch (error) { // handle the error } ```

| Parameter | Presence | Description | |---------|--------|-----------| **placementId** | required | The identifier of the Placement. This is the value you specified when creating a placement in your Adapty Dashboard. | | **locale** |

optional

default: `en`
```

|

The identifier of the paywall localization. This parameter is expected to be a language code composed of one or more subtags separated by the minus (-) character. The first subtag is for the language, the second one is for the region.

Example: `en` means English, `pt-br` represents the Brazilian Portuguese language.

See Localizations and locale codes for more information on locale codes and how we recommend using them.

| | **fetchPolicy** | default: `.reloadRevalidatingCacheData` |

By default, SDK will try to load data from the server and will return cached data in case of failure. We recommend this variant because it ensures your users always get the most up-to-date data.

However, if you believe your users deal with unstable internet, consider using `.returnCacheDataElseLoad` to return cached data if it exists. In this scenario, users might not get the absolute latest data, but they'll experience faster loading times, no matter how patchy their internet connection is. The cache is updated regularly, so it's safe to use it during the session to avoid network requests.

Note that the cache remains intact upon restarting the app and is only cleared when the app is reinstalled or through manual cleanup.

Adapty SDK stores paywalls in two layers: regularly updated cache described above and fallback paywalls . We also use CDN to fetch paywalls faster and a stand-alone fallback server in case the CDN is unreachable. This system is designed to make sure you always get the latest version of your paywalls while ensuring reliability even in cases where internet connection is scarce.

| | **loadTimeout** | default: 5 sec |

This value limits the timeout for this method. If the timeout is reached, cached data or local fallback will be returned.

Note that in rare cases this method can timeout slightly later than specified in `loadTimeout`, since the operation may consist of different requests under the hood.

|

Don't hardcode product IDs! Since paywalls are configured remotely, the available products, the number of products, and special offers (such as free trials) can change over time. Make sure your code handles these scenarios.
For example, if you initially retrieve 2 products, your app should display those 2 products. However, if you later retrieve 3 products, your app should display all 3 without requiring any code changes. The only thing you have to hardcode is placement ID.

Response parameters:

| Parameter | Description | | :-------- | :---------- | | Paywall | An `AdaptyPaywall` object with: a list of pro IDs, the paywall identifier, remote config, and several other properties. |

Fetch products

Once you have the paywall, you can query the product array that corresponds to it:

```
swift Adapty.getPaywallProducts(paywall: paywall) { result in switch result { case let .success(products): // the requested products array case let .failure(error): // handle the error } } kotlin Adapty.getPaywallProducts(paywall) { result -> when (result) { is AdaptyResult.Success -> { val products = result.value // the requested products } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getPaywallProducts(paywall, result -> { if (result instanceof AdaptyResult.Success) { List products = ((AdaptyResult.Success>) result).getValue(); // the requested products

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final products = await Adapty().getPaywallProducts(paywall: paywall); // the requested products array } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.GetPaywallProducts(paywall, (products, error) => { if(error != null) { // handle the error return; }

// products - the requested products array }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { // ...paywall const products = await adapty.getPaywallProducts(paywall); // the requested products list } catch (error) { // handle the error } ```
```

Response parameters:

| Parameter | Description | | :-------- | :---------- | | Products | List of `AdaptyPaywallProduct` objects with: product identifier, product name, price, currency, subscription length, and several other properties. |

Check intro offer eligibility on iOS

After getting products and before presenting the paywall, you might want to check if the user qualifies for an introductory offer for an iOS subscription and handle cases where they don't qualify. On iOS, this usually means examining different factors like whether the user is new to the subscription or has already used an introductory offer for it.

You don't have to manually check these factors on iOS. Moreover, if you use the Paywall Builder, you can skip the eligibility check as it will be done automatically. However, if you do not use the Paywall Builder, use the `getProductsIntroductoryOfferEligibility(products:)` method. It automatically checks the eligibility status for each product in the array:

```
swift Adapty.getProductsIntroductoryOfferEligibility(products: products) { result in switch result { case .success(let eligibilities): // update your UI case let .failure(error): // handle the error } } javascript try { final eligibilities = await Adapty().getProductsIntroductoryOfferEligibility(products: products); // update your UI } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } csharp Adapty.GetProductsIntroductoryOfferEligibility(products, (eligibilities, error) => { if (eligibilities != null) { // update your UI } if (error != null { // handle the error } });
```

Next, you can see all the possible values of `AdaptyEligibility`

| Value | Descriptions | | :---------- | :---------- | | eligible | The user is eligible for an intro offer, it is safe to reflect this info in your UI. | | ineligible | The user is no eligible to get any offer, you shouldn't present it in your UI. | | notApplicable | This product is not configured to have an offer. |

:::warning We urge you to be very careful with this scenario, as Apple's reviewers can check it quite rigorously. However, based on our experience with them, we conclude that the behavior of the payment environment in which they perform their checks may be somewhat different from our usual sandbox and production. :::

Speed up paywall fetching with default audience paywall

Typically, paywalls are fetched almost instantly, so you donâ€™t need to worry about speeding up this process. However, in cases where you have numerous audiences and paywalls, and your users have a weak internet connection, fetching a paywall may take longer than you'd like. In such situations, you might want to display a default paywall to ensure a smooth user experience rather than showing no paywall at all.

To address this, you can use the `getPaywallForDefaultAudience` method, which fetches the paywall of the specified placement for the **All Users** audience. However, it's crucial to understand that the recommended approach is to fetch the paywall by the `getPaywall` method, as detailed in the Fetch Paywall Information section above.

:::warning Why we recommend using `getPaywall`

The `getPaywallForDefaultAudience` method comes with a few significant drawbacks:

- **Potential backward compatibility issues**: If you need to show different paywalls for different app versions (current and future), you may face challenges. Youâ€™ll either have to design paywalls that support the current (legacy) version or accept that users with the current (legacy) version might encounter issues with non-rendered paywalls.
- **Loss of targeting**: All users will see the same paywall designed for the **All Users** audience, which means you lose personalized targeting (including based on countries, marketing attribution or your own custom attributes).

If you're willing to accept these drawbacks to benefit from faster paywall fetching, use the `getPaywallForDefaultAudience` method as follows. Otherwise stick to `getPaywall` described above :::

```
swift Adapty.getPaywallForDefaultAudience(placementId: "YOUR_PLACEMENT_ID", locale: "en") { result in switch result { case let .success(paywall): // the requested paywall case let .failure(error): // handle the error } } kotlin Adapty.getPaywallForDefaultAudience("YOUR_PLACEMENT_ID", locale = "en") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java Adapty.getPaywallForDefaultAudience("YOUR_PLACEMENT_ID", "en", result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue(); // the requested paywall

} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error

}

}); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const id = 'YOUR_PLACEMENT_ID'; const locale = 'en';

const paywall = await adapty.getPaywallForDefaultAudience(id, locale);
```

// the requested paywall } catch (error) { // handle the error } ```

:::note The `getPaywallForDefaultAudience` method is available starting from these versions:

- iOS: 2.11.2
- Android: 2.11.3
- React Native: 2.11.2

The method is not yet supported in Flutter and Unity, but support will be added soon. :::

| Parameter | Presence | Description | |---------|--------|-----------| **placementId** | required | The identifier of the Placement. This is the value you specified when creating a placement in your Adapty Dashboard. | | **locale** |

optional

default: `en`

|

The identifier of the paywall localization. This parameter is expected to be a language code composed of one or more subtags separated by the minus (-) character. The first subtag is for the language, the second one is for the region.

Example: `en` means English, `pt-br` represents the Brazilian Portuguese language.

See Localizations and locale codes for more information on locale codes and how we recommend using them.

| | **fetchPolicy** | default: `.reloadRevalidatingCacheData` |

By default, SDK will try to load data from the server and will return cached data in case of failure. We recommend this variant because it ensures your users always get the most up-to-date data.

However, if you believe your users deal with unstable internet, consider using `.returnCacheDataElseLoad` to return cached data if it exists. In this scenario, users might not get the absolute latest data, but they'll experience faster loading times, no matter how patchy their internet connection is. The cache is updated regularly, so it's safe to use it during the session to avoid network requests.

Note that the cache remains intact upon restarting the app and is only cleared when the app is reinstalled or through manual cleanup.

|

---

title: "Render paywall designed by remote config" description: "Discover how to implement and display a custom paywall and remote config paywall in your mobile app, ensuring flexibility and control over its content and appearance, while checking user eligibility for introductory offers."

metadataTitle: "How to Render a Custom Paywall and Remore Config Paywall"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

If you've customized a paywall using remote config, you'll need to implement rendering in your mobile app's code to display it to users. Since remote config offers flexibility tailored to your needs, you're in control of what's included and how your paywall view appears. We provide a method for fetching the remote configuration, giving you the autonomy to showcase your custom paywall configured via remote config.

Don't forget to check if a user is eligible for an introductory offer in iOS and adjust the paywall view to process the case when they are eligible.

Get paywall remote config and present it

To get a remote config of a paywall, access the `remoteConfig` property and extract the needed values.

```
swift Adapty.getPaywall("YOUR_PLACEMENT_ID") { result in let paywall = try? result.get() let headerText = paywall?.remoteConfig?.dictionary?["header_text"] as? String } kotlin
Adapty.getPaywall("YOUR_PLACEMENT_ID") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value val headerText =
paywall.remoteConfig?.dataMap?.get("header_text") as? String } is AdaptyResult.Error -> { val error = result.error // handle the error } } } ```java
Adapty.getPaywall("YOUR_PLACEMENT_ID", result -> { if (result instanceof AdaptyResult.Success) { AdaptyPaywall paywall = ((AdaptyResult.Success) result).getValue();

    AdaptyPaywall.RemoteConfig remoteConfig = paywall.getRemoteConfig();

    if (remoteConfig != null) {
        if (remoteConfig.getDataMap().get("header_text") instanceof String) {
            String headerText = (String) remoteConfig.getDataMap().get("header_text");
        }
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final paywall = await Adapty().getPaywall(id: "YOUR_PLACEMENT_ID"); final String? headerText = paywall.remoteConfig?['header_text']; } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } ```
```

At this point, once you've received all the necessary values, it's time to render and assemble them into a visually appealing page. Ensure that the design accommodates various mobile phone screens and orientations, providing a seamless and user-friendly experience across different devices.

:::warning Make sure to record the paywall view event as described below, allowing Adapty analytics to capture information for funnels and A/B tests. :::

After you've done with displaying the paywall, continue with setting up a purchase flow. When the user makes a purchase, simply call `.makePurchase()` with the product from your paywall. For details on the `.makePurchase()` method, read Making purchases.

We recommend creating a backup paywall called a fallback paywall. This backup will display to the user when there's no internet connection or cache available, ensuring a smooth experience even in these situations.

Track paywall view events

Adapty assists you in measuring the performance of your paywalls. While we gather data on purchases automatically, logging paywall views needs your input because only you know when a customer sees a paywall.

To log a paywall view event, simply call `.logShowPaywall(paywall)`, and it will be reflected in your paywall metrics in funnels and A/B tests.

```
swift Adapty.logShowPaywall(paywall) kotlin Adapty.logShowPaywall(paywall) java Adapty.logShowPaywall(paywall); javascript try { final result = await
Adapty().logShowPaywall(paywall: paywall); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } csharp Adapty.LogShowPaywall(paywall, (error) => { // handle
the error }); typescript await adapty.logShowPaywall(paywall);
```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **paywall** | required | An `AdaptyPaywall` object. |

---

title: "Make purchases in mobile app" description: "Learn how to seamlessly integrate paywalls into your mobile app and enable purchases using Adapty. Discover the .makePurchase() method for making purchases. Explore how Adapty automatically applies promotional offers to enhance user transactions."

metadataTitle: "How to Make Purchases in Code for Paywalls in Adapty"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Displaying paywalls within your mobile app is an essential step in offering users access to premium content or services. However, simply presenting these paywalls is enough to support purchases only if you use Paywall Builder to customize your paywalls.

If you don't use the Paywall Builder, you must use a separate method called `.makePurchase()` to complete a purchase and unlock the desired content. This method serves as the gateway for users to engage with the paywalls and proceed with their desired transactions.

If your paywall has an active promotional offer for the product a user is trying to buy, Adapty will automatically apply it at the time of purchase.

:::warning Keep in mind that the introductory offer will be applied automatically only if you use the paywalls set up using the Paywall Builder.

In other cases, you'll need to verify the user's eligibility for an introductory offer on iOS. Skipping this step may result in your app being rejected during release. Moreover, it could lead to charging the full price to users who are eligible for an introductory offer. :::

Make sure you've done the initial configuration without skipping a single step. Without it, we can't validate purchases.

Make purchase

:::note In paywalls built with Paywall Builder purchases are processed automatically with no additional code. If that's your case â€” you can skip this step. :::

```
swift Adapty.makePurchase(product: product) { result in switch result { case let .success(info): if info.profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive ?? false { // successful purchase } case let .failure(error): // handle the error } } ```kotlin Adapty.makePurchase(activity, product) { result -> when (result) { is AdaptyResult.Success -> { val info = result.value //NOTE: info is null in case of cross-grade with DEFERRED proration mode val profile = info?.profile

        if (profile?.accessLevels?.get("YOUR_ACCESS_LEVEL")?.isActive == true) {
            // grant access to premium features
        }
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
    }
}

} </TabItem> <TabItem value="java" label="Java" default> java Adapty.makePurchase(activity, product, result -> { if (result instanceof AdaptyResult.Success) { AdaptyPurchasedInfo info = ((AdaptyResult.Success) result).getValue(); //NOTE: info is null in case of cross-grade with DEFERRED proration mode AdaptyProfile profile = info != null ? info.getProfile() : null;

    if (profile != null) {
        AdaptyProfile.AccessLevel premium = profile.getAccessLevels().get("YOUR_ACCESS_LEVEL");

        if (premium != null && premium.isActive()) {
            // successful purchase
        }
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final profile = await Adapty().makePurchase(product: product); if (profile?.accessLevels['YOURACCESSLEVEL']?.isActive ?? false) { // successful purchase
} } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.MakePurchase(product, (profile, error) => { if(error != null) { // handle error return; }

var accessLevel = profile.AccessLevels["YOURACCESSLEVEL"]; if (accessLevel != null && accessLevel.IsActive) { // grant access to features } }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const profile = await adapty.makePurchase(product); const isSubscribed = profile?.accessLevels['YOURACCESSLEVEL']?.isActive;

if (isSubscribed) {
    // grant access to features in accordance with access level
}

} catch (error) { // handle the error } ```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Product** | required | An AdaptyPaywallProduct object retrieved from the payw

Response parameters:

| Parameter | Description | |---------|----------| |**Profile** |

An AdaptyProfile object provides comprehensive information about a user's access levels, subscriptions, and non-subscription purchases within the app.

Check the access level status to ascertain whether the user has the required access to the app.

|

Below is a complete example of making the purchase of the access level premium. Premium is the default access level, so in most cases, your code will look this way:

```
swift Adapty.makePurchase(product: product) { result in switch result { case let .success(info): if info.profile.accessLevels["premium"]?.isActive ?? false { // grant access to premium features } case let .failure(error): // handle the error } } ```kotlin Adapty.makePurchase(activity, product) { result -> when (result) { is AdaptyResult.Success -> { val info = result.value //NOTE: info is null in case of cross-grade with DEFERRED proration mode val profile = info?.profile

        if (profile?.accessLevels?.get("premium")?.isActive == true) {
            // grant access to premium features
        }
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
    }
}

} </TabItem> <TabItem value="java" label="Java" default> java Adapty.makePurchase(activity, product, result -> { if (result instanceof AdaptyResult.Success) { AdaptyPurchasedInfo info = ((AdaptyResult.Success) result).getValue(); //NOTE: info is null in case of cross-grade with DEFERRED proration mode AdaptyProfile profile = info != null ? info.getProfile() : null;

    if (profile != null) {
        AdaptyProfile.AccessLevel premium = profile.getAccessLevels().get("premium");

        if (premium != null && premium.isActive()) {
            // grant access to premium features
        }
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final profile = await Adapty().makePurchase(product: product); if (profile?.accessLevels['premium']?.isActive ?? false) { // grant access to premium features
} } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.MakePurchase(product, (profile, error) => { if(error != null) { // handle error return; }

// "premium" is an identifier of default access level var accessLevel = profile.AccessLevels["premium"]; if (accessLevel != null && accessLevel.IsActive) { // grant access to premium features } }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const profile = await adapty.makePurchase(product); const isSubscribed = profile?.accessLevels['premium']?.isActive;

if (isSubscribed) {
    // grant access to premium features
}

} catch (error) { // handle the error } ```

:::warning **Note:** if you're still on Apple's StoreKit version lower than v2.0 and Adapty SDK version lowers than v.2.9.0, you need to provide Apple App Store shared secret instead. This method is currently deprecated by Apple. :::

Change subscription when making a purchase

When a user opts for a new subscription instead of renewing the current one, the way it works depends on the app store:

- For the App Store, the subscription is automatically updated within the subscription group. If a user purchases a subscription from one group while already having a subscription from another, both subscriptions will be active at the same time.
- For Google Play, the subscription isn't automatically updated. You'll need to manage the switch in your mobile app code as described below.

To replace the subscription with another one in Android, call .makePurchase() method with the additional parameter:

```kotlin Adapty.makePurchase(activity, product, subscriptionUpdateParams) { result -> when (result) { is AdaptyResult.Success -> { val info = result.value //NOTE: info is null in case of cross-grade with DEFERRED proration mode val profile = info?.profile

        // successful cross-grade
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
```

```
    }
}
```

} </TabItem> <TabItem value="java" label="Java" default> java Adapty.makePurchase(activity, product, subscriptionUpdateParams, result -> { if (result instanceof AdaptyResult.Success) { AdaptyPurchasedInfo info = ((AdaptyResult.Success) result).getValue(); //NOTE: info is null in case of cross-grade with DEFERRED proration mode AdaptyProfile profile = info != null ? info.getProfile() : null;

```
    // successful cross-grade
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final result = await adapty.makePurchase(product: product, subscriptionUpdateParams: subscriptionUpdateParams,);

// successful cross-grade } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.MakePurchase(product, subscriptionUpdateParams, (profile, error) => { if(error != null) { // handle error return; }

// successful cross-grade }); ```

```
typescript // TODO: add example
```

Additional request parameter:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **subscriptionUpdateParams** | required | an `AdaptySubscriptionUpdateParameters` object. |

You can read more about subscriptions and proration modes in the Google Developer documentation:

- About subscriptions
- Recommendations from Google for proration modes
- Proration mode `IMMEDIATE_AND_CHARGE_PRORATED_PRICE`. Note: this method is available only for subscription upgrades. Downgrades are not supported.
- Proration mode `DEFERRED`. Note: in case of success, `profile` in the callback will be returned as `null` since a real subscription change will occur only when the current subscription's billing period ends.

Make a deferred purchase in iOS

For deferred purchases on iOS, Adapty SDK has an optional delegate method, which is called when the user starts the purchase in the App Store, and the transaction continues in your app. Just store `makeDeferredPurchase` and call it later if you want to hold your purchase for now. Then show the paywall to your user. To continue purchase, call `makeDeferredPurchase`.

```swift title="Swift" extension AppDelegate: AdaptyDelegate { func paymentQueue(shouldAddStorePaymentFor product: AdaptyDeferredProduct, defermentCompletion makeDeferredPurchase: @escaping (ResultCompletion?) -> Void) { // you can store makeDeferredPurchase callback and call it later

```
    // or you can call it right away
    makeDeferredPurchase { result in
        // check the purchase
    }
}
```

} ```

## Redeem Offer Code in iOS

Since iOS 14.0, your users can redeem Offer Codes. Code redemption means using a special code, like a promotional or gift card code, to get free access to content or features in an app or on the App Store. To enable users to redeem offer codes, you can display the offer code redemption sheet by using the appropriate SDK method:

```
swift Adapty.presentCodeRedemptionSheet() typescript adapty.presentCodeRedemptionSheet();
```

:::danger Based on our observations, the Offer Code Redemption sheet in some apps may not work reliably. We recommend redirecting the user directly to the App Store.

In order to do this, you need to open the url of the following format: `https://apps.apple.com/redeem?ctx=offercodes&id={apple_app_id}&code={code}` :::

---

title: "Restore purchases in mobile app" description: "Learn how to implement the restore purchases feature in your iOS and Android apps using Adapty. Discover the importance of enabling users to regain access to their previously purchased content without additional charges, and explore the simple process of restoring purchases using the .restorePurchases() method."

metadataTitle: "Adapty: How to Restore Purchases on iOS and Android Apps"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Restoring Purchases in both iOS and Android is a feature that allows users to regain access to previously purchased content, such as subscriptions or in-app purchases, without being charged again. This feature is especially useful for users who may have uninstalled and reinstalled the app or switched to a new device and want to access their previously purchased content without paying again.

:::note In paywalls built with Paywall Builder, purchases are restored automatically without additional code from you. If that's your case â€" you can skip this step. :::

To restore a purchase if you do not use the Paywall Builder to customize the paywall, call `.restorePurchases()` method:

```
swift Adapty.restorePurchases { [weak self] result in switch result { case let .success(profile): if info.profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive ?? false { // successful access restore } case let .failure(error): // handle the error } } ```kotlin Adapty.restorePurchases { result -> when (result) { is AdaptyResult.Success -> { val profile = result.value

        if (profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive == true) {
        // successful access restore
        }
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
    }
}
```

} </TabItem> <TabItem value="java" label="Java" default> java Adapty.restorePurchases(result -> { if (result instanceof AdaptyResult.Success) { AdaptyProfile profile = ((AdaptyResult.Success) result).getValue();

```
    if (profile != null) {
        AdaptyProfile.AccessLevel premium = profile.getAccessLevels().get("YOUR_ACCESS_LEVEL");

        if (premium != null && premium.isActive()) {
            // successful access restore
        }
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript try { final profile = await Adapty().restorePurchases(); if (profile?.accessLevels['YOUR*ACCESS*LEVEL']?.isActive ?? false) { // successful access restore
} } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.RestorePurchases((profile, error) => { if (error != null) { // handle the error return; }

var accessLevel = profile.AccessLevels["YOUR*ACCESS*LEVEL"]; if (accessLevel != null && accessLevel.IsActive) { // restore access } }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { const profile = await adapty.restorePurchases(); const isSubscribed = profile.accessLevels['YOUR*ACCESS*LEVEL']?.isActive;

```
if (isSubscribed) {
    // restore access
}
```

} catch (error) { // handle the error } ```

Response parameters:

| Parameter | Description | |---------|----------| |**Profile** |

An `AdaptyProfile` object. This model contains info about access levels, subscriptions, and non-subscription purchases.

Đ¡heck the **access level status** to determine whether the user has access to the app.
```

|

---

title: "Check subscription status" description: "Easily check subscription status by retrieving active access levels from the AdaptyProfile object in Adapty. Stay updated on changes made in it"

metadataTitle: "Check Subscription Status with AdaptyProfile Access Levels"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem'; import Details from '@site/src/components/Details';

Adapty makes it easy to track subscription status without needing to manually input product IDs into your code. Instead, you can simply check for an active access level to confirm a user's subscription status.

You can create multiple access levels for a single app. For example, in a newspaper app, you might sell subscriptions to different topics like "sports" and "science." However, most apps only require one access level. If thatâ€™s the case for your app, you can use the default "premium" access level.

Once you determine the user's subscription status (access level), you can grant access to the appropriate features in your app.

▶ Before you start checking subscription status (Click to Expand)

- For iOS, set up App Store Server Notifications
- For Android, set up Real-time Developer Notifications (RTDN)

Access level and the AdaptyProfile object

Access levels are properties of the AdaptyProfile object. We recommend retrieving the profile when your app starts, such as when you identify a user , and then updating it whenever changes occur. This way, you can use the profile object without repeatedly requesting it.

To be notified of profile updates, listen for profile changes as described in the Listening for profile updates, including access levels section below.

Retrieving the access level from the server

To get the access level from the server, use the `.getProfile()` method:

`swift Adapty.getProfile { result in if let profile = try? result.get() { // check the access profile.accessLevels["YOUR_ACCESS_LEVEL"]?.isActive ?? false { // grant access to premium features } } } kotlin Adapty.getProfile { result -> when (result) { is AdaptyResult.Success -> { val profile = result.value // check the access } is AdaptyResult.Error -> { val error = result.error // handle the error } } }` ```java Adapty.getProfile(result -> { if (result instanceof AdaptyResult.Success) { AdaptyProfile profile = ((AdaptyResult.Success) result).getValue(); // check the access

```
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript try { final profile = await Adapty().getProfile(); // check the access } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp Adapty.GetProfile((profile, error) => { if (error != null) { // handle the error return; }

// check the access }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript try { const profile = await adapty.getProfile(); } catch (error) { // handle the error } ```

Response parameters:

| Parameter | | | :--------- | :---------- | | **Profile** | An Ada object. You generally need to check only the access level status of the profile to determine if the user has premium access to the app. The `.getProfile` method provides the most up-to-date result by querying the API. If the Adapty SDK fails to retrieve information due to reasons like no internet connection, cached data will be returned. The Adapty SDK regularly updates the `AdaptyProfile` cache to keep the information as current as possible. |

Example: Checking the default "premium" access level

`swift Adapty.getProfile { result in if let profile = try? result.get(), profile.accessLevels["premium"]?.isActive ?? false { // grant access to premium features } }` ```kotlin Adapty.getProfile { result -> when (result) { is AdaptyResult.Success -> { val profile = result.value

```
        if (profile.accessLevels["premium"]?.isActive == true) {
            // grant access to premium features
        }
    }
    is AdaptyResult.Error -> {
        val error = result.error
        // handle the error
    }
}
```

} `</TabItem> <TabItem value="java" label="Java" default>` java Adapty.getProfile(result -> { if (result instanceof AdaptyResult.Success) { AdaptyProfile profile = ((AdaptyResult.Success) result).getValue();

```
    AdaptyProfile.AccessLevel premium = profile.getAccessLevels().get("premium");

    if (premium != null && premium.isActive()) {
        // grant access to premium features
    }
} else if (result instanceof AdaptyResult.Error) {
    AdaptyError error = ((AdaptyResult.Error) result).getError();
    // handle the error
}
```

}); `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript try { final profile = await Adapty().getProfile(); if (profile?.accessLevels['premium']?.isActive ?? false) { // grant access to premium features } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp Adapty.GetProfile((profile, error) => { if (error != null) { // handle the error return; }

// "premium" is an identifier of default access level var accessLevel = profile.AccessLevels["premium"]; if (accessLevel != null && accessLevel.IsActive) { // grant access to premium features } }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript try { const profile = await adapty.getProfile();

const isActive = profile.accessLevels["premium"]?.isActive; if (isActive) { // grant access to premium features } } catch (error) { // handle the error } ```

Listening for profile updates, including access levels

Adapty fires an event whenever the userâ€™s profile is updated.

To receive profile updates, including changes to subscription status (access levels), follow these steps:

```swift Adapty.delegate = self

// To receive user profile updates, extend `AdaptyDelegate` with this method: func didLoadLatestProfile(_ profile: AdaptyProfile) { // handle any changes to user profile } `</TabItem> <TabItem value="kotlin" label="Kotlin" default>` kotlin Adapty.setOnProfileUpdatedListener { profile -> // handle any changes to user profile } `</TabItem> <TabItem value="java" label="Java" default>` java Adapty.setOnProfileUpdatedListener(profile -> { // handle any changes to user profile }); `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript Adapty().didUpdateProfileStream.listen((profile) { // handle any changes to user profile }); `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp // Extend `AdaptyEventListener` with `OnLoadLatestProfile` method: public class AdaptyListener : MonoBehaviour, AdaptyEventListener { public void OnLoadLatestProfile(Adapty.Profile profile) { // handle any changes to suser profile } } `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript // Create an "onLatestProfileLoad" event listener adapty.addEventListener('onLatestProfileLoad', profile => { // handle any changes to user profile }); ```

Adapty also triggers an event when the application starts. In this case, the cached profile will be returned.

## Caching profile including access level

The Adapty SDK includes a cache that stores the user profile and access level as a part of it. This ensures that even if the server is unavailable, you can still access the profileâ€™s subscription status. However, note that you cannot directly request data from the cache. The SDK regularly queries the server every minute to check for updates or changes related to the profile. Any modifications, such as new transactions, will be synced with the cached data to keep it aligned with the server

---

title: "Identify users" description: ""
```

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty creates an internal profile ID for every user. However, if you have your own authentication system, you should set your own Customer User ID. You can find users by their Customer User ID in the Profiles section and use it in the server-side API, which will be sent to all integrations.

Setting customer user ID on configuration

If you have a user ID during configuration, just pass it as `customerUserId` parameter to `.activate()` method:

```
swift Adapty.activate("PUBLIC_SDK_KEY", customerUserId: "YOUR_USER_ID") kotlin Adapty.activate(applicationContext, "PUBLIC_SDK_KEY", customerUserId = "YOUR_USER_ID") java Adapty.activate(getApplicationContext(), "PUBLIC_SDK_KEY", observerMode, "YOUR_USER_ID"); typescript adapty.activate("PUBLIC_SDK_KEY", { customerUserId: "YOUR_USER_ID" });
```

You may notice that there are no snippets for Flutter and Unity. Unfortunately, there are technical limitations that won't allow passing the ID upon activation.

Setting customer user ID after configuration

If you don't have a user ID in the SDK configuration, you can set it later at any time with the `.identify()` method. The most common cases for using this method are after registration or authorization, when the user switches from being an anonymous user to an authenticated user.

```
swift Adapty.identify("YOUR_USER_ID") { error in if error == nil { // successful identify } } kotlin Adapty.identify("YOUR_USER_ID") { error -> if (error == null) { // successful identify } } java Adapty.identify("YOUR_USER_ID", error -> { if (error == null) { // successful identify } }); javascript try { await Adapty().identify(customerUserId); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } csharp Adapty.Identify("YOUR_USER_ID", (error) => { if(error == null) { // successful identify } }); typescript try { await adapty.identify("YOUR_USER_ID"); // successfully identified } catch (error) { // handle the error }
```

Request parameters:

- **Customer User ID** (required): a string user identifier.

:::warning Resubmitting of significant user data

In some cases, such as when a user logs into their account again, Adapty's servers already have information about that user. In these scenarios, the Adapty SDK will automatically switch to work with the new user. If you passed any data to the anonymous user, such as custom attributes or attributions from third-party networks, you should resubmit that data for the identified user.

It's also important to note that you should re-request all paywalls and products after identifying the user, as the new user's data may be different. :::

Logging out and logging in

You can logout the user anytime by calling `.logout()` method:

```
swift Adapty.logout { error in if error == nil { // successful logout } } kotlin Adapty.logout { error -> if (error == null) { // successful logout } } java Adapty.logout(error -> { if (error == null) { // successful logout } }); javascript try { await Adapty().logout(); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } csharp Adapty.Logout((error) => { if(error == null) { // successful logout } }); typescript try { await adapty.logout(); // successful logout } catch (error) { // handle the error }
```

You can then login the user using `.identify()` method.

---

title: "Set user attributes" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

You can set optional attributes such as email, phone number, etc, to the user of your app. You can then use attributes to create user segments or just view them in CRM.

Setting user attributes

To set user attributes, call `.updateProfile()` method:

```swift let builder = AdaptyProfileParameters.Builder() .with(email: "email@email.com") .with(phoneNumber: "+18888888888") .with(facebookAnonymousId: "facebookAnonymousId") .with(amplitudeUserId: "amplitudeUserId") .with(amplitudeDeviceId: "amplitudeDeviceId") .with(mixpanelUserId: "mixpanelUserId") .with(appmetricaProfileId: "appmetricaProfileId") .with(appmetricaDeviceId: "appmetricaDeviceId") .with(firstName: "John") .with(lastName: "Appleseed") .with(gender: .other) .with(birthday: Date())

Adapty.updateProfile(params: builder.build()) { error in if error != nil { // handle the error
} } </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin val builder = AdaptyProfileParameters.Builder() .withEmail("email@email.com") .withPhoneNumber("+18888888888") .withFacebookAnonymousId("facebookAnonymousId") .withAmplitudeUserId("amplitudeUserId") .withAmplitudeDeviceId("amplitudeDeviceId") .withMixpanelUserId("mixpanelUserId") .withAppmetricaProfileId("appmetricaProfileId") .withAppmetricaDeviceId("appmetricaDeviceId") .withFirstName("John") .withLastName("Appleseed") .withGender(AdaptyProfile.Gender.OTHER) .withBirthday(AdaptyProfile.Date(1970, 1, 3))

Adapty.updateProfile(builder.build()) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="java" label="Java" default> java AdaptyProfileParameters.Builder builder = new AdaptyProfileParameters.Builder() .withEmail("email@email.com") .withPhoneNumber("+18888888888") .withFacebookAnonymousId("facebookAnonymousId") .withAmplitudeUserId("amplitudeUserId") .withAmplitudeDeviceId("amplitudeDeviceId") .withMixpanelUserId("mixpanelUserId") .withAppmetricaProfileId("appmetricaProfileId") .withAppmetricaDeviceId("appmetricaDeviceId") .withFirstName("John") .withLastName("Appleseed") .withGender(AdaptyProfile.Gender.OTHER) .withBirthday(new AdaptyProfile.Date(1970, 1, 3));

Adapty.updateProfile(builder.build(), error -> { if (error != null) { // handle the error } }); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript final builder = AdaptyProfileParametersBuilder() ..setEmail("email@email.com") ..setPhoneNumber("+18888888888") ..setFacebookAnonymousId("facebookAnonymousId") ..setAmplitudeUserId("amplitudeUserId") ..setAmplitudeDeviceId("amplitudeDeviceId") ..setMixpanelUserId("mixpanelUserId") ..setAppmetricaProfileId("appmetricaProfileId") ..setAppmetricaDeviceId("appmetricaDeviceId") ..setFirstName('John') ..setLastName('Appleseed') ..setGender(AdaptyProfileGender.other) ..setBirthday(DateTime(1970, 1, 3));

try { await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> typescript // Only for TypeScript validation import type { AdaptyProfileParameters } from 'react-native-adapty';

const params: AdaptyProfileParameters = { email: 'email@email.com', phoneNumber: '+18888888888', facebookAnonymousId: 'facebookAnonymousId', amplitudeUserId: 'amplitudeUserId', amplitudeDeviceId: 'amplitudeDeviceId', mixpanelUserId: 'mixpanelUserId', appmetricaProfileId: 'appmetricaProfileId', appmetricaDeviceId: 'appmetricaDeviceId', firstName: 'John', lastName: 'Appleseed', gender: 'other', birthday: new Date().toISOString(), };

try { await adapty.updateProfile(params); } catch (error) { // handle AdaptyError } </TabItem> <TabItem value="RN" label="React Native (TS)" default> csharp var builder = new Adapty.ProfileParameters.Builder() .SetFirstName("John") .SetLastName("Appleseed") .SetBirthday(new DateTime(1970, 1, 3)) .SetGender(ProfileGender.Female) .SetEmail("example@adapty.io");

Adapty.UpdateProfile(builder.Build(), (error) => { if(error != nil) { // handle the error
} }); ```

Please note that the attributes that you've previously set with the `updateProfile` method won't be reset.

### The allowed keys list

The allowed keys `&lt;Key&gt;` of `AdaptyProfileParameters.Builder` and the values `&lt;Value&gt;` are listed below:

| Key | Value | |---|-----| |

email

phoneNumber

facebookAnonymousId

amplitudeUserId

amplitudeDeviceId

mixpanelUserId

appmetricaProfileId

appmetricaDeviceId

firstName
```

lastName

| String up to 30 characters | | gender | Enum, allowed values are: `female`, `male`, `other` | | birthday | Date |

App Tracking Transparency Status (starting iOS 14)

If your application uses AppTrackingTransparency framework and presents an app-tracking authorization request to the user, then you should send the authorization status to Adapty.

```swift if #available(iOS 14, macOS 11.0, *) { let builder = AdaptyProfileParameters.Builder() .with(appTrackingTransparencyStatus: .authorized)

Adapty.updateProfile(params: builder.build()) { [weak self] error in
    if error != nil {
        // handle the error
    }
}

} </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript final builder = AdaptyProfileParametersBuilder()
..setAppTrackingTransparencyStatus(AdaptyIOSAppTrackingTransparencyStatus.authorized);

try { await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp var builder = new Adapty.ProfileParameters.Builder(); .SetAppTrackingTransparencyStatus(IOSAppTrackingTransparencyStatus.Authorized);

Adapty.UpdateProfile(builder.Build(), (error) => { if(error != nil) { // handle the error
} }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import {AppTrackingTransparencyStatus} from 'react-native-adapty';

try { await adapty.updateProfile({ // you can also pass a string value (validated via tsc) if you prefer appTrackingTransparencyStatus: AppTrackingTransparencyStatus.Authorized, }); } catch (error) { // handle AdaptyError } ```

:::warning We strongly recommend that you send this value as early as possible when it changes, only in that case the data will be sent in a timely manner to the integrations you have configured. :::

### Custom user attributes

You can set your own custom attributes. These are usually related to your app usage. For example, for fitness applications, they might be the number of exercises per week, for language learning app user's knowledge level, and so on. You can use them in segments to create targeted paywalls and offers, and you can also use them in analytics to figure out which product metrics affect the revenue most.

swift do { builder = try builder.with(customAttribute: "value1", forKey: "key1") } catch { // handle key/value validation error } kotlin builder.withCustomAttribute("key1", "value1") java builder.withCustomAttribute("key1", "value1"); ```javascript try { final builder = AdaptyProfileParametersBuilder() ..setCustomStringAttribute('value1', 'key1') ..setCustomDoubleAttribute(1.0, 'key2');

await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp try { builder = builder.SetCustomStringAttribute("string*key", "string*value"); builder = builder.SetCustomDoubleAttribute("double*key", 123.0f); } catch (Exception e) { // handle the exception } </TabItem> <TabItem value="RN" label="React Native (TS)" default> *typescript try { await adapty.updateProfile({ codableCustomAttributes: { key*1: 'value*1', *key*2: 2, }, }); } catch (error) { // handle AdaptyError } ```

To remove existing key, use `.withRemoved(customAttributeForKey:)` method:

swift do { builder = try builder.withRemoved(customAttributeForKey: "key2") } catch { // handle error } kotlin builder.withRemovedCustomAttribute("key2") java builder.withRemovedCustomAttribute("key2"); ```javascript try { final builder = AdaptyProfileParametersBuilder() ..removeCustomAttribute('key1') ..removeCustomAttribute('key2');

await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp try { builder = builder.RemoveCustomAttribute("key*to*remove"); } catch (Exception e) { // handle the exception } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript try { // to remove a key, pass null as its value await adapty.updateProfile({ codableCustomAttributes: { key*1: null, *key*2: null, }, }); } catch (error) { // handle AdaptyError } ```

Sometimes you need to figure out what custom attributes have already been installed before. To do this, use the `customAttributes` field of the `AdaptyProfile` object.

:::warning Keep in mind that the value of `customAttributes` may be out of date since the user attributes can be sent from different devices at any time so the attributes on the server might have been changed after the last sync. :::

### Limits

You can set up to 30 custom attributes per user, with key names up to 30 characters long and values up to 50 characters long.

- Up to 30 custom attributes per user
- Key name up to 30 characters long. The key name can include alphanumeric characters and any of the following: `_` `-` `.`
- Value up to 50 characters long

---

title: "Track onboarding screens" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

The onboarding stage is a very common situation in modern mobile apps. The quality of its implementation, content, and number of steps can have a rather significant influence on further user behavior, especially on his desire to become a subscriber or simply make some purchases.

In order for you to be able to analyze user behavior at this critical stage without leaving Adapty, we have implemented the ability to send dedicated events every time a user visits yet another onboarding screen.

To do this, simply call the `.logShowOnboarding` function:

swift Adapty.logShowOnboarding(name: "onboarding_name", screenName: "first_screen", screenOrder: 1) kotlin Adapty.logShowOnboarding(name = "onboarding_name", screenName = "first_screen", screenOrder = 1) java Adapty.logShowOnboarding("onboarding_name", "first_screen", 1); javascript try { await Adapty().logShowOnboarding(name: 'onboarding_name', screenName: 'first_screen', screenOrder: 1); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } csharp Adapty.LogShowOnboarding("onboarding_name", "first_screen", 1, (error) => { if(error != null) { // handle the error } }); typescript await adapty.logShowOnboarding( 1, /* screenOrder */ 'onboarding_name', /* name */ 'first_screen' /* screenName */ );

| Parameter | Presence | Description | | :-------------- | :------- | :---------- | | **name** | optional | The name of your onboarding | | **screenNa**
optional | The readable name of a particular screen as part of onboarding | | **screenOrder** | required | An unsigned integer value representing the order of this screen in your onboarding sequence (it must be greater than 0) |

:::warning Even though there is only one mandatory parameter in this function, we recommend that you think of names for all the screens, as this will make the work of analysts during the data examination phase much easier. :::

---

title: "Use fallback paywalls" description: ""

## metadataTitle: ""

A paywall is an in-app storefront where customers can see and purchase products within your mobile app. Typically, paywalls are fetched from the server when a customer accesses them. However, Adapty allows you to have fallback paywalls for situations when a user opens the app without a connection to the Adapty backend (e.g., no internet connection or in the rare case of backend unavailability) and there's no cache on the device.

Adapty generates fallbacks as a JSON file in the necessary format, reflecting English versions of the paywalls you've configured in the Adapty Dashboard. Download it and pass it or its contents to the `Adapty.setFallbackPaywalls` method, following the instructions specific per framework:

- iOS
- Android
- Flutter
- React Native
- Unity

---

title: "iOS - Use fallback paywalls" description: ""

## metadataTitle: ""

To use fallback paywalls:

1. Place the fallback JSON file you downloaded in the Adapty Dashboard alongside your app in the user's device.
2. Call the `.setFallbackPaywalls` method. Place this method in your code **before** fetching a paywall, ensuring that the mobile app possesses it when a fallback paywall is required to replace the standard one.
```

Here's an example of retrieving fallback paywall data from a locally stored JSON file named `ios_fallback.json`.

```swift title="Swift" if let url = Bundle.main.url(forResource: "ios_fallback", withExtension: "json") { Adapty.setFallbackPaywalls(fileURL: url)

```

Parameters:

| Parameter | Description | | :---------- | :---------- | | **fileURL** | Path to the file with fallback payw you downloaded in the Adapty Dashboard. |

---

title: "Android - Use fallback paywalls" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

To use fallback paywalls:

1. Place the fallback JSON file you downloaded in the Adapty Dashboard alongside your app in the user's device.
2. Call the `.setFallbackPaywalls` method. Place this method in your code **before** fetching a paywall, ensuring that the mobile app possesses it when a fallback paywall is required to replace the standard one.

Here's an example of retrieving fallback paywall data from locally stored JSON file named `android_fallback.json`.

``kotlin //if you put the 'android_fallback.json' file to the 'assets' directory val location = FileLocation.fromAsset("android_fallback.json") //orFileLocation.fromAsset("/android_fallback.json")` if you placed it in a child folder of 'assets')

//if you put the 'android*fallback.json' file to the 'res/raw' directory val location = FileLocation.fromResId(context, R.raw.android*fallback)

//pass the file location Adapty.setFallbackPaywalls(location, callback) `</TabItem> <TabItem value="java" label="Java" default>` java //if you put the 'android*fallback.json' file to the 'assets' directory FileLocation location = FileLocation.fromAsset("android*fallback.json"); //or `FileLocation.fromAsset("<additional_folder>/android_fallback.json");` if you placed it in a child folder of 'assets')

//if you put the 'android*fallback.json' file to the 'res/raw' directory FileLocation location = FileLocation.fromResId(context, R.raw.android*fallback);

//pass the file location Adapty.setFallbackPaywalls(location, callback); ```

Parameters:

| Parameter | Description | | :---------- | :---------- | | **location** | The FileLocation for the file with fallback paywalls |

Alternatively, you can use a URI instead of the file location. here is an example of how to do so:

```kotlin val fileUri: Uri = //get Uri for the file with fallback paywalls // for example, if you put the 'android*fallback.json' file to 'res/raw' directory, // you can obtain the Uri as follows: // // val fileUri = Uri.Builder() // .scheme(ContentResolver.SCHEME*ANDROID*RESOURCE) // .authority(yourAuthority) //usually your applicationId // .appendPath("${R.raw.android*fallback}") // .build()

Adapty.setFallbackPaywalls(fileUri, callback) `</TabItem> <TabItem value="java" label="Java" default>` java Uri fileUri = //get Uri for the file with fallback paywalls // for example, if you put the 'android*fallback.json' file to 'res/raw' directory, // you can obtain the Uri as follows: // // Uri fileUri = new Uri.Builder() // .scheme(ContentResolver.SCHEME*ANDROID*RESOURCE) // .authority(yourAuthority) //usually your applicationId // .appendPath(String.valueOf(R.raw.android*fallback)) // .build();

Adapty.setFallbackPaywalls(fileUri, callback); ```

Parameters:

| Parameter | Description | | :---------- | :---------- | | **fileUri** | The Uri for the file with fallback paywalls |

---

title: "Flutter - Use fallback paywalls" description: ""

## metadataTitle: ""

To use fallback paywalls, call the `.setFallbackPaywalls` method. Pass the content of the fallback JSON file you downloaded in the Adapty Dashboard. Place this method in your code **before** fetching a paywall, ensuring that the mobile app possesses it when a fallback paywall is required to replace the standard one.

```javascript title="Flutter" import 'dart:async' show Future; import 'dart:io' show Platform; import 'package:flutter/services.dart' show rootBundle;

final filePath = Platform.isIOS ? 'assets/ios*fallback.json' : 'assets/android*fallback.json'; final jsonString = await rootBundle.loadString(filePath);

try { await adapty.setFallbackPaywalls(jsonString); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } ```

Parameters:

| Parameter | Description | | :---------- | :---------- | | **jsonString** | The contents of the fallback JS file you downloaded in the Adapty Dashboard. |

---

title: "React Native - Use fallback paywalls" description: ""

## metadataTitle: ""

Follow the instructions below to use the fallback paywalls in your mobile app code.

### For Android

1. Place the fallback file you downloaded in the Adapty Dashboard to a directory on the native layer. There are 2 correct directories to put the file: `android/app/src/main/assets/` or `android/app/src/main/res/raw/`.
   Please keep in mind that the `res/raw` folder has a special file naming convention (start with a letter, no capital letters, no special characters except for the underscore, and no spaces in the names).
   1. **For android/app/src/main/assets/**: Pass the file path relatively to the `assets` directory, for example:
      - `{ relativeAssetPath: 'android_fallback.json' }` if you placed the file to the root of `assets` itself
      - `{ relativeAssetPath: '<additional_folder>/android_fallback.json' }` if you placed it in a child folder of `assets`
   2. **For android/app/src/main/res/raw/**: Pass `{ rawResName: 'android_fallback' }`. Type the file name without the file extension.
2. Pass the result of step 2 to the `android` property of `FallbackPaywallsLocation`.

### For iOS

1. In XCode, use the menu **File** -> **Add Files to "YourProjectName"** to add the fallback file you downloaded in the Adapty Dashboard.
2. Pass `{ fileName: 'ios_fallback.json' }` to the `ios` property of `FallbackPaywallsLocation`.

Here's an example of retrieving fallback paywall data from locally stored JSON files named `android_fallback.json` and `ios_fallback.json`.

`typescript title="Current (v2.11+)" //after v2.11 const paywallsLocation = { ios: { fileName: 'ios_fallback.json' }, android: { //if the file is located in 'android/app/src/main/assets/' relativeAssetPath: 'android_fallback.json' } } await adapty.setFallbackPaywalls(paywallsLocation);` ```typescript title="Legacy (before v2.11)" //Legacy (before v2.11) const fallbackPaywalls = Platform.select({ ios: require('./ios*fallback.json'), android: require('./android*fallback.json'), }); // React Native automatically parses JSON, but we do not need that const fallbackString = JSON.stringify(fallbackPaywalls);

await adapty.setFallbackPaywalls(fallbackString); ```

Parameters:

| Parameter | Description | | :---------- | :---------- | | **paywallsLocation** | The object represents the location of the file resource. |

---

title: "Unity - Use fallback paywalls" description: ""

## metadataTitle: ""

To use fallback paywalls, call the `.setFallbackPaywalls` method. Pass the content of the fallback JSON file you downloaded in the Adapty Dashboard. Place this method in your code **before** fetching a paywall, ensuring that the mobile app possesses it when a fallback paywall is required to replace the standard one.

```
csharp title="Unity" Adapty.SetFallbackPaywalls("<FALLBACK_PAYWALL_DATA>", (error) => { if(error != null) { // handle error } });
```

Parameters:

| Parameter | Presence | Description | | :---------- | :------ | :---------- | | <FALLBACK*PAYWALL*DATA> | required | The contents of the fallback JSON file you downloaded in the Adapty Dashboard |

---

title: "Use localizations and locale codes" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Why this is important

There are a few scenarios when locale codes come into play â€” for example, when you're trying to fetch the correct paywall for the current localization of your app.

As locale codes are complicated and can vary from platform to platform, we rely on an internal standard for all the platforms we support. However, because these codes are complicated, it is really important for you to understand what exactly are you sending to our server to get the correct localization, and what happens next â€” so you will always receive what you expect.

Locale code standard at Adapty

For locale codes, Adapty uses a slightly modified BCP 47 standard: every code consists of lowercase subtags, separated by hyphens. Some examples: `en` (English), `pt-br` (Portuguese (Brazil)), `zh` (Simplified Chinese), `zh-hant` (Traditional Chinese).

Locale code matching

When Adapty receives a call from the client-side SDK with the locale code and starts looking for a corresponding localization of a paywall, the following happens:

1. The incoming locale string is converted to lowercase and all the underscores (_) are replaced with hyphens (-)
2. We then look for the localization with the fully matching locale code
3. If no match was found, we take the substring before the first hyphen (`pt` for `pt-br`) and look for the matching localization
4. If no match was found again, we return the default `en` localization

This way an iOS device that sent `'pt_BR'`, an Android device that sent `pt-BR`, and another device that sent `pt-br` will get the same result.

Implementing localizations: recommended way

If you're wondering about localizations, chances are you're already dealing with the localized string files in your project. If that's the case, we recommend placing some key-value with the intended Adapty locale code in each of your files for the corresponding localizations. And then extract the value for this key when calling our SDK, like so:

```swift // 1. Modify your Localizable.strings files

/* Localizable.strings - Spanish */ "adapty_paywalls_locale" = "es"; /* Localizable.strings - Portuguese (Brazil) */ "adapty_paywalls_locale" = "pt-br"; // 2. Extract and use the locale code let locale = NSLocalizedString("adapty_paywalls_locale", comment: "") // pass locale code to AdaptyUI.getViewConfiguration or Adapty.getPaywall method </TabItem> <TabItem value="kotlin" label="Android" default> kotlin // 1. Modify your strings.xml files

/* strings.xml - Spanish */ es

/* strings.xml - Portuguese (Brazil) */ pt-br

// 2. Extract and use the locale code

val localeCode = context.getString(R.string.adapty_paywalls_locale) // pass locale code to AdaptyUI.getViewConfiguration or Adapty.getPaywall method ```

That way you can ensure you're in full control of what localization will be retrieved for every user of your app.

## Implementing localizations: the other way

You can get similar (but not identical) results without explicitly defining locale codes for every localization. That would mean extracting a locale code from some other objects that your platform provides, like this:

```
swift let locale = Locale.current.identifier // pass locale code to AdaptyUI.getViewConfiguration or Adapty.getPaywall method ```kotlin val locale = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) context.resources.configuration.locales[0] else context.resources.configuration.locale

val localeCode = locale.toLanguageTag() // pass locale code to AdaptyUI.getViewConfiguration or Adapty.getPaywall method ```
```

Note that we don't recommend this approach due to few reasons:

1. On iOS preferred languages and current locale are not identical. If you want the localization to be picked correctly you'll have to either rely on Apple's logic, which works out of the box if you're using the recommended approach with localized string files, or re-create it.
2. It's hard to predict what exactly will Adapty's server get. For example, on iOS, it is possible to obtain a locale like `ar_OM@numbers='latn'` on a device and send it to our server. And for this call you will get not the `ar-om` localization you were looking for, but rather `ar`, which is likely unexpected.

Should you decide to use this approach anyway â€” make sure you've covered all the relevant use cases.

---

title: "Implement Observer mode" description: ""

## metadataTitle: ""

If you already have your own purchase infrastructure and aren't prepared to fully switch to Adapty, you can explore Observer mode. It'll provide you with the best analytics capabilities, integration with attribution and analytics systems, and a CRM with your users' profiles.

To make all this work in Observer mode, it's enough to enable it when configuring the Adapty SDK by setting the `observerMode` parameter to `true` as described in Adapty SDK configuration for iOS, Android, Flutter, React Native, and Unity.

If you want to also use Adapty's paywalls and A/B test functionality, it is possible, though it will require extra effort on your part in Observer mode compared to Full mode. To do this:

1. Depending on whether you designed your paywalls with Paywall Builder or remote config, the flow will be slightly different:
   - Paywalls designed with remote config may contain various data as it allows for any kind of JSON content. However, in this case, you're the one responsible for interpreting that data and rendering your paywall. Implementation of remote config paywalls does not differ from Full mode, so do it as described in Display paywalls designed with remote config for Full mode.
   - Paywalls designed with Paywall Builder contain both what should be shown within the paywall and how it should be shown. Implementation of Paywall Builder paywalls differs for Observer mode. Refer to the guides for iOS and Android for detailed guidance. This is necessary for implementing purchases outside of Paywall Builder paywalls, which typically handle purchases themselves.
2. Implement the purchasing process according to your requirements.
3. Associate used paywalls to purchase transactions for Adapty to determine the source of purchases.

---

title: "iOS - Present Paywall Builder paywalls in Observer mode" description: ""

## metadataTitle: ""

import Details from '@site/src/components/Details';

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

:::note This section refers to Observer mode only. If you do not work in Observer mode, refer to the iOS - Present Paywall Builder paywalls. :::

▶ Before you start presenting paywalls (Click to Expand)
```

1. Set up initial integration of Adapty with the Google Play and with the App Store.
2. Install and configure Adapty SDK. Make sure to set the `observerMode` parameter to `true`. Refer to our framework-specific instructions for iOS, for Android, for Flutter, for React Native, and for Unity.
3. Create products in the Adapty Dashboard.
4. Configure paywalls, assign products to them, and customize them using Paywall Builder in the Adapty Dashboard.
5. Create placements and assign your paywalls to them in the Adapty Dashboard.
6. Fetch Paywall Builder paywalls and their configuration in your mobile app code.

Present Paywall Builder paywalls in Swift

1. Implement the `AdaptyObserverModeDelegate` object:

   ```
   swift title="Swift" func paywallController(_ controller: AdaptyPaywallController, didInitiatePurchase product: AdaptyPaywallProduct, onStartPurchase: @escaping () -> Void, onFinishPurchase: @escaping () -> Void) { // use the product object to handle the purchase // use the onStartPurchase and onFinishPurchase callbacks to notify AdaptyUI about the process of the purchase }
   ```

   The `paywallController(_:didInitiatePurchase:onStartPurchase:onFinishPurchase:)` event will inform you that the user has initiated a purchase. You can trigger your custom purchase flow in response to this event.

   Also, remember to invoke the following callbacks to notify AdaptyUI about the process of the purchase. This is necessary for proper paywall behavior, such as showing the loader, among other things:

   | Callback | Description | | :---------- | :---------- | | onStartPurchase | The callback should be invoked to notify AdaptyUI that the purchase is started. | | onFinishPurchase | The callback should be invoked to notify AdaptyUI that the purchase is finished. |

2. Initialize the visual paywall you want to display by using the `.paywallController(for:products:viewConfiguration:delegate:observerModeDelegate:)` method:

   ```swift title="Swift" import Adapty import AdaptyUI

   let visualPaywall = AdaptyUI.paywallController( for: , products: , viewConfiguration: , delegate: observerModeDelegate: ) ```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Paywall** | required | An `AdaptyPaywall` object to obtain a controller for the desired paywall. | | **Products** | optional | Provide an array of `AdaptyPaywallProducts` to optimize the display timing of products on the screen. If `nil` is passed, AdaptyUI will automatically fetch the necessary products. | | **ViewConfiguration** | required | An `AdaptyUI.LocalizedViewConfiguration` object containing visual details of the paywall. Use the `AdaptyUI.getViewConfiguration(paywall:locale:)` method. Refer to Fetch Paywall Builder paywalls and their configuration topic for more details. | | **Delegate** | required | An `AdaptyPaywallControllerDelegate` to listen to paywall events. Refer to Handling paywall events topic for more details. | | **ObserverModeDelegate** | required | The `AdaptyObserverModeDelegate` object you've implemented in the previous step | | **TagResolver** | optional | Define a dictionary of custom tags and their resolved values. Custom tags serve as placeholders in the paywall content, dynamically replaced with specific strings for personalized content within the paywall. Refer to Custom tags in paywall builder topic for more details. |

Returns:

| Object | Description | | :---------- | :---------- | | AdaptyPaywallController | An object, representing the requested paywall screen |

After the object has been successfully created, you can display it like so:

```
swift title="Swift" present(visualPaywall, animated: true)
```

:::warning Don't forget to Associate paywalls to purchase transactions. Otherwise, Adapty will not determine the source paywall of the purchase. :::

## Present Paywall Builder paywalls in SwiftUI

In order to display the visual paywall on the device screen, use the `.paywall` modifier in SwiftUI:

```swift title="SwiftUI" @State var paywallPresented = false

var body: some View { Text("Hello, AdaptyUI!") .paywall( isPresented: $paywallPresented, paywall: , configuration: , didPerformAction: { action in switch action { case .close: paywallPresented = false default: // Handle other actions break } }, didFinishRestore: { profile in /* check access level and dismiss */ }, didFailRestore: { error in /* handle the error */ }, didFailRendering: { error in paywallPresented = false }, observerModeDidInitiatePurchase: { product, onStartPurchase, onFinishPurchase in // use the product object to handle the purchase // use the onStartPurchase and onFinishPurchase callbacks to notify AdaptyUI about the process of the purchase }, ) } ```

Request parameters:

| Parameter | Presence | Description | | :---------- | :------- | :---------- | | **Paywall** | required | An `AdaptyPaywall` object to obtain a controller for the desired paywall. | | **Product** | optional | Provide an array `AdaptyPaywallProducts` to optimize the display timing of products on the screen. If `nil` is passed, AdaptyUI will automatically fetch the necessary products. | | **Configuration** | required | An `AdaptyUI.LocalizedViewConfiguration` object containing visual details of the paywall. Use the `AdaptyUI.getViewConfiguration(paywall:locale:)` method. Refer to Fetch Paywall Builder paywalls and their configuration topic for more details. | | **TagResolver** | optional | Define a dictionary of custom tags and their resolved values. Custom tags serve as placeholders in the paywall content, dynamically replaced with specific strings for personalized content within the paywall. Refer to Custom tags in paywall builder topic for more details. |

Closure parameters:

| Closure parameter | Description | | :---------- | :---------- | | **didFinishRestore** | If Adapty.restorePurchases() succeeds, this callback will be invok | | **didFailRestore** | If Adapty.restorePurchases() fails, this callback will be invoked. | | **didFailRendering** | If an error occurs during the interface rendering, this callback will be invoked. | | **observerModeDidInitiatePurchase** | This callback is invoked when a user initiates a purchase. |

Refer to the iOS - Handling events topic for other closure parameters.

:::warning Don't forget to Associate paywalls to purchase transactions. Otherwise, Adapty will not determine the source paywall of the purchase. :::

---

title: "Android - Present Paywall Builder paywalls in Observer mode" description: ""

## metadataTitle: ""

import Details from '@site/src/components/Details'; import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

If you've customized a paywall using the Paywall Builder, you don't need to worry about rendering it in your mobile app code to display it to the user. Such a paywall contains both what should be shown within the paywall and how it should be shown.

:::note This section refers to Observer mode only. If you do not work in Observer mode, refer to the Android - Present Paywall Builder paywalls topic instead. :::

▶ Before you start presenting paywalls (Click to Expand)

1. Set up initial integration of Adapty with the Google Play and with the App Store.
2. Install and configure Adapty SDK. Make sure to set the `observerMode` parameter to `true`. Refer to our framework-specific instructions for iOS, for Android, for Flutter, for React Native, and for Unity.
3. Create products in the Adapty Dashboard.
4. Configure paywalls, assign products to them, and customize them using Paywall Builder in the Adapty Dashboard.
5. Create placements and assign your paywalls to them in the Adapty Dashboard.
6. Fetch Paywall Builder paywalls and their configuration in your mobile app code.
7. Implement the `AdaptyUiObserverModeHandler`. The `AdaptyUiObserverModeHandler`'s callback (`onPurchaseInitiated`) informs you when the user initiates a purchase. You can trigger your custom purchase flow in response to this callback like this:

```
kotlin val observerModeHandler = AdaptyUiObserverModeHandler { product, paywall, paywallView, onStartPurchase, onFinishPurchase -> onStartPurchase() yourBillingClient.makePurchase( product, onSuccess = { purchase -> onFinishPurchase() //handle success }, onError = { onFinishPurchase() //handle error }, onCancel = { onFinishPurchase() //handle cancel } ) } java AdaptyUiObserverModeHandler observerModeHandler = (product, paywall, paywallView, onStartPurchase, onFinishPurchase) -> { onStartPurchase.invoke(); yourBillingClient.makePurchase( product, purchase -> { onFinishPurchase.invoke(); //handle success }, error -> { onFinishPurchase.invoke(); //handle error }, () -> { //cancellation onFinishPurchase.invoke(); //handle cancel } ); };
```

Also, remember to invoke these callbacks to AdaptyUI. This is necessary for proper paywall behavior, such as showing the loader, among other things:

| Callback in Kotlin | Callback in Java | Description | | :---------- | :---------- | :---------- | | onStartPurchase() | onStartPurchase.invoke() | The callback should be invoked to notify AdaptyUI that the purchase is started. | | onFinishPurchase() | onFinishPurchase.invoke() | The callback should be invoked to notify AdaptyUI that the purchase is finished successfully or not, or the purchase is canceled. |

2. In order to display the visual paywall, you must first initialize it. To do this, call the method `AdaptyUI.getPaywallView()` or create the `AdaptyPaywallView` directly:

```kotlin val paywallView = AdaptyUI.getPaywallView( activity, viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), eventListener, personalizedOfferResolver, tagResolver, observerModeHandler, )

//======= OR =======

val paywallView = AdaptyPaywallView(activity) // or retrieve it from xml ... with(paywallView) { setEventListener(eventListener) setObserverModeHandler(observerModeHandler) showPaywall( viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), personalizedOfferResolver, tagResolver, ) } </TabItem> <TabItem value="java" label="Java" default> java AdaptyPaywallView paywallView = AdaptyUI.getPaywallView( activity, viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), eventListener, personalizedOfferResolver, tagResolver, observerModeHandler );

//======= OR =======

AdaptyPaywallView paywallView = new AdaptyPaywallView(activity); //add to the view hierarchy if needed, or you receive it from xml ... paywallView.setEventListener(eventListener); paywallView.setObserverModeHandler(observerModeHandler); paywallView.showPaywall(viewConfiguration, products, AdaptyPaywallInsets.of(topInset, bottomInset), personalizedOfferResolver); </TabItem> <TabItem value="XML" label="XML" default> xml width="matchparent" android:layout*height="match*parent" /> ```

After the view has been successfully created, you can add it to the view hierarchy and display it.

Request parameters:

| Parameter | Presence | Description | |---------|--------|-----------| **Products** | optional | Provide an array of `AdaptyPaywallProduct`to optimize the display timing of products on the screen. If `null` is passed, AdaptyUI will automatically fetch the required products. | | **ViewConfiguration** | required | Supply an `AdaptyViewConfiguration` object containing visual details of the paywall. Use the `Adapty.getViewConfiguration(paywall)` method to load it. Refer to Fetch the visual configuration of paywall topic for more details. | | **Insets** | required | Define an `AdaptyPaywallInsets` object containing information about the area overlapped by system bars, creating vertical margins for content. If neither the status bar nor the navigation bar overlaps the `AdaptyPaywallView`, pass `AdaptyPaywallInsets.NONE`. For fullscreen mode where system bars overlap part of your UI, obtain insets as shown under the table. | | **EventListener** | optional | Provide an `AdaptyUiEventListener` to observe paywall events. Extending AdaptyUiDefaultEventListener is recommended for ease of use. Refer to Handling paywall events topic for more details. | | **PersonalizedOfferResolver** | optional | To indicate personalized pricing (read more ), implement `AdaptyUiPersonalizedOfferResolver` and pass your own logic that maps `AdaptyPaywallProduct` to true if the product's price is personalized, otherwise false. | | **TagResolver** | optional | Use `AdaptyUiTagResolver` to resolve custom tags within the paywall text. This resolver takes a tag parameter and resolves it to a corresponding string. Refer to Custom tags in paywall builder topic for more details. | | **ObserverModeHandler** | required for Observer mode | The `AdaptyUiObserverModeHandler` you've implemented in the previous step. | | **variationId** | required | The string identifier of the variation. You can get it using `variationId` property of the `AdaptyPaywall` object. | | **transaction** | required |

For iOS, StoreKit1: an `SKPaymentTransaction` object.

For iOS, StoreKit 2: Transaction object.

For Android: String identifier (`purchase.getOrderId()`) of the purchase, where the purchase is an instance of the billing library Purchase class.

|

For fullscreen mode where system bars overlap part of your UI, obtain insets in the following way:

```kotlin import androidx.core.graphics.Insets import androidx.core.view.ViewCompat import androidx.core.view.WindowInsetsCompat

//create extension function fun View.onReceiveSystemBarsInsets(action: (insets: Insets) -> Unit) { ViewCompat.setOnApplyWindowInsetsListener(this) { _, insets -> val systemBarInsets = insets.getInsets(WindowInsetsCompat.Type.systemBars()) ViewCompat.setOnApplyWindowInsetsListener(this, null) action(systemBarInsets) insets } } //and then use it with the view paywallView.onReceiveSystemBarsInsets { insets -> val paywallInsets = AdaptyPaywallInsets.of(insets.top, insets.bottom) paywallView.setEventListener(eventListener) paywallView.setObserverModeHandler(observerModeHandler) paywallView.showPaywall(viewConfig, products, paywallInsets, personalizedOfferResolver, tagResolver) } </TabItem> <TabItem value="java" label="Java" default> java import androidx.core.graphics.Insets; import androidx.core.view.ViewCompat; import androidx.core.view.WindowInsetsCompat;

...

ViewCompat.setOnApplyWindowInsetsListener(paywallView, (view, insets) -> { Insets systemBarInsets = insets.getInsets(WindowInsetsCompat.Type.systemBars()); ViewCompat.setOnApplyWindowInsetsListener(paywallView, null);

```
   AdaptyPaywallInsets paywallInsets =
            AdaptyPaywallInsets.of(systemBarInsets.top, systemBarInsets.bottom);
   paywallView.setEventListener(eventListener);
   paywallView.setObserverModeHandler(observerModeHandler);
   paywallView.showPaywall(viewConfiguration, products, paywallInsets, personalizedOfferResolver, tagResolver);

   return insets;
```

}); ```

Returns:

| Object | Description | | :----------------- | :-- | | `AdaptyPaywallView` | object, representing the requested paywall screen. |

:::warning Don't forget to Associate paywalls to purchase transactions. Otherwise, Adapty will not determine the source paywall of the purchase. :::

---

title: "Associate paywalls to purchase transactions in Observer mode" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

In Observer mode, Adapty SDK cannot determine the source of purchases as you are the one processing them. Therefore, if you intend to use paywalls and/or A/B tests in Observer mode, you need to associate the transaction coming from your app store with the corresponding paywall in your mobile app code. This is important to get right before releasing your app, otherwise it will lead to errors in analytics.

After the preliminary configuration is done, you need to associate the transaction generated by Apple or Google to the corresponding paywall in your mobile app code using the `variationId` parameter as shown in the example below. Make sure you always associate the transaction with the paywall that generated it, only then you will be able to see the correct metrics in the Adapty Dashboard.

```swift let variationId = paywall.variationId

// There are two overloads: for StoreKit 1 and StoreKit 2 Adapty.setVariationId(variationId, forPurchasedTransaction: transaction) { error in if error == nil { // successful binding } } </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin Adapty.setVariationId(transactionId, variationId) { error -> if (error == null) { // success } } </TabItem> <TabItem value="java" label="Java" default> java Adapty.setVariationId(transactionId, variationId, error -> { if (error == null) { // success } }); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript final transactionId = transaction.transactionIdentifier final variationId = paywall.variationId

try { await Adapty().setVariationId('transaction_id', variationId); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp Adapty.SetVariationForTransaction("", "", (error) => { if(error != null) { // handle the error return; }

```
// successful binding
```

}); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript const variationId = paywall.variationId;

try { await adapty.setVariationId('transaction_id', variationId); } catch (error) { // handle the `AdaptyError` } ```

Request parameters:

| Parameter | Presence | Description | |---------|--------|-----------| **variationId** | required | The string identifier of the variation. You can get it using `variationId` property of the `AdaptyPaywall` object. | | **transaction** | required |

For iOS, StoreKit1: an `SKPaymentTransaction` object.

For iOS, StoreKit 2: Transaction object.

For Android: String identifier (`purchase.getOrderId()`) of the purchase, where the purchase is an instance of the billing library Purchase class.

|

For accurate analytics, ensure the transaction is associated with the paywall within 3 hours of its creation.

---

title: "API reference" description: ""

## metadataTitle: ""

- iOS SDK
- Android SDK
- Flutter SDK
```

---

title: "SDK Models" description: ""

metadataTitle: ""

Interfaces

AdaptyPaywallProduct

An information about a product.

| Name | Type | Description | | :---------- | :---------- | :---------- | | vendorProductId | string | Unique identifier of a product from App Store Connect or Google Play Console | | introductoryOfferEligibility | er
| User's eligibility for your introductory offer. Check this property before displaying info about introductory offers (i.e. free trials) | | promotionalOfferEligibility (*iOS only*) | boolean | User's eligibility for the promotional offers. Check this property before displaying info about promotional offers | | promotionalOfferId (*iOS only*) | string (Optional) | For iOS: An identifier of a promotional offer, provided by Adapty for this specific user. No value for Android devices | | paywallABTestName | string | Parent A/B test name | | paywallName | string | Parent paywall name | | skProduct (*iOS only*) | SKProduct (iOS) | Underlying system representation of the product | | skuDetails (*Android only*) | SkuDetails assigned to this product | Underlying system representation of the product | | currencyCode | string (optional) | The currency code of the locale used to format the price of the product. | | currencySymbol | string (optional) | The currency symbol of the locale used to format the price of the product. | | discounts (*iOS only*) | [`AdaptyProductDiscount`] | An array of subscription offers available for the auto-renewable subscription. (Will be empty for iOS version below 12.2 and macOS version below 10.14.4). | | introductoryDiscount | `AdaptyProductDiscount` (optional) | The object containing introductory price information for the product. (Will be nil for iOS version below 11.2 and macOS version below 10.14.4). | | isFamilyShareable (*iOS only*) | bool | A Boolean value that indicates whether the product is available for family sharing in App Store Connect. (Will be false for iOS version below 14.0 and macOS version below 11.0). | | localizedDescription | string | A description of the product. | | localizedPrice | string (optional for ios) | The priceâ€™s language is determined by the preferred language set on the device. | | localizedSubscriptionPeriod | string (optional) | The periodâ€™s language is determined by the preferred language set on the device. | | localizedTitle | string | The name of the product. | | price | number | The cost of the product in the local currency. | | regionCode (*iOS only*) | string (optional) | The region code of the locale used to format the price of the product. | | subscriptionGroupIdentifier (*iOS only*) | string (optional) | The identifier of the subscription group to which the subscription belongs. (Will be nil for iOS version below 12.0 and macOS version below 10.14). | | subscriptionPeriod | `AdaptyProductSubscriptionPeriod` (optional) | The period details for products that are subscriptions. (Will be nil for iOS version below 11.2 and macOS version below 10.14.4). |

AdaptyProductSubscriptionPeriod

| Name | Type | Description | | :---------- | :---------- | :---------- | | unit | AdaptyPeriodUnit | A unit of time that a subscripti
period is specified in. The possible values are: `day`, `week`, `month`, `year` and `unknown` | | numberOfUnits | number | A number of period units |

AdaptyProductDiscount

An information about a product discount.

| Name | Type | Description | |----|----|----------| | identifier (*iOS only*) | string (Optional) | Unique identifier of a discount offer for a product | | price | number | Discount price of a product in a local currency | | numberOfPeriods | number | A number of periods this product discount is available | | paymentMode (*iOS only*) | string |

For iOS: A payment mode for this product discount. Possible values are `freeTrial`, `payUpFront`, `payAsYouGo`

No value for Android devices

| | localizedPrice | string (Optional) | A formatted price of a discount for a user's locale | | localizedSubscriptionPeriod | string (Optional for iOS) | A formatted subscription period of a discount for a user's locale. | | localizedNumberOfPeriods (*iOS only*) | string (Optional) | For iOS: A formatted number of periods of a discount for a user's locale | | subscriptionPeriod | `AdaptyProductSubscriptionPeriod` | An information about period for a product discount |

AdaptyPaywall

An information about a paywall.

| Name | Type | Description | |----|----|----------| | id | string | An identifier of a paywall, configured in Adapty Dashboard | | variationId | string | An identifier of a variation, used to attribute purchases to this paywall | | revision | number | Current revision (version) of a paywall. Every change within a paywall creates a new revision | | remoteConfigString | string (optional) | A custom JSON string configured in Adapty Dashboard for this paywall | | remoteConfig | dictionary (optional) | A custom dictionary configured in Adapty Dashboard for this paywall (same as `remoteConfigString`) | | vendorProductIds | array of strings | Array of related products ids | | abTestName | string | Parent A/B test name | | name | string | Paywall name | | locale | string |

An identifier of a paywall locale

This parameter is expected to be a language code composed of one or more subtags separated by the "-" character. The first subtag is for the language, the second one is for the region (The support for regions will be added later).

Example: `en` means English, `en-US` represents US English.

If the parameter is omitted, the paywall will be returned in the default locale.

|

AdaptyProfile

An information about a user's subscription status and purchase history.

| Name | Type | Description | | :---------- | :---------- | :---------- | | profileId | string | An identifier of a user in Adapty | | customerUserId | string (Optional) | An identifier of a user in your system | | customAttributes | dictionary | Previously set user custom attributes with `.updateProfile()` method | | accessLevels | dictionary<string, `AccessLevel`> | The keys are access level identifiers configured by you in Adapty Dashboard. The values are Can be null if the customer has no access levels | | subscriptions | dictionary<string, `Subscription`> | The keys are product ids from a store. The values are information about subscriptions. Can be null if the customer has no subscriptions | | nonSubscriptions | dictionary<string, [`NonSubscription`]> | The keys are product ids from the store. The values are arrays of information about consumables. Can be null if the customer has no purchases. |

AdaptyProfile.AccessLevel

Information about the user's access level.

| Name | Type | Description | |----|----|----------| | id | string | Unique identifier of the access level configured by you in Adapty Dashboard | | isActive | boolean | True if this access level is active. Generally, you can check this property to determine wether a user has an access to premium features | | vendorProductId | string | An identifier of a product in a store that unlocked this access level | | store | string | A store of the purchase that unlocked this access level. Possible values are `'app_store' | 'play_store' | 'adapty'` | | activatedAt |

iOS: Date

Android: string (ISO 8601 datetime)

| Time when this access level was activated. | | startsAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when this access level has started (could be in the future). | | renewedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the access level was renewed. | | expiresAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the access level will expire (could be in the past and could be null for lifetime access). | | isLifetime | boolean | True if this access level is active for a lifetime (no expiration date) | | willRenew | boolean | True

if this auto-renewable subscription is set to renew | | isInGracePeriod | boolean | True if this auto-renewable subscription is in the grace period | | unsubscribedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the auto-renewable subscription was cancelled. Subscription can still be active, it just means that auto-renewal turned off. Will be set to null if the user reactivates the subscription. | | billingIssueDetectedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when billing issue was detected. Subscription can still be active. Would be set to null if a charge is made. | | cancellationReason | string (Optional) | A reason why a subscription was cancelled. Possible values are `'voluntarily_cancelled' | 'billing_error' | 'refund' | 'price_increase' | 'product_was_not_available' | 'unknown'` | | isRefund | boolean | True if this purchase was refunded | | activeIntroductoryOfferType | string (Optional) | A type of an active introductory offer. If the value is not null, it means that the offer was applied during the current subscription period. Possible values: `'free_trial' | 'pay_as_you_go' | 'pay_up_front'` | | activePromotionalOfferType | string (Optional) | A type of an active promotional offer. If the value is not null, it means that the offer was applied during the current subscription period. Possible values `'free_trial' | 'pay_as_you_go' | 'pay_up_front'` | | activePromotionalOfferId | string (Optional) | An id of active promotional offer. |

AdaptyProfile.Subscription

Information about the user's subscription.

| Name | Type | Description | |----|----|----------| | store | string | A store of the purchase that unlocked this subscription. Possible values:`'app_store' | 'play_store' | 'adapty'` | | vendorProductId | string | An identifier of a product in a store that unlocked this subscription | | vendorTransactionId | string | A transaction id of a purchase in a store that unlocked this subscription | | vendorOriginalTransactionId | string | An original transaction id of the purchase in a store that unlocked this subscription. For auto-renewable subscription, this will be an id of the first transaction in this subscription | | isActive | boolean | True if the subscription is active | | isLifetime | boolean | True if the subscription is active for a lifetime (no expiration date) | | activatedAt |

iOS: Date

Android: string (ISO 8601 datetime)

| Time when the subscription was activated. | | renewedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the subscription was renewed. | | expiresAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the subscription will expire (could be in the past and could be null for a lifetime access). | | startsAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the subscription has started (could be in the future). | | unsubscribedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when the auto-renewable subscription was cancelled. Subscription can still be active, it means that auto-renewal is turned off. Would be null if a user reactivates the subscription | | billingIssueDetectedAt |

iOS: Date

Android: string (ISO 8601 datetime)

(optional)

| Time when a billing issue was detected. Subscription can still be active | | willRenew | boolean | True if the auto-renewable subscription is set to renew | | isInGracePeriod | boolean | Whether the auto-renewable subscription is in a grace period | | cancellationReason | string (Optional) | A reason why the subscription was cancelled. Possible values: `'voluntarily_cancelled' | 'billing_error' | 'refund' | 'price_increase' | 'product_was_not_available' | 'unknown'` | | isRefund | bool | True if the purchase was refunded | | activeIntroductoryOfferType | string (Optional) | A type of an active introductory offer. If the value is not null, it means that the offer was applied during the current subscription period. Possible values: `'free_trial' | 'pay_as_you_go' | 'pay_up_front'` | | activePromotionalOfferType | string (Optional) | A type of an active promotional offer. If the value is not null, it means that the offer was applied during the current subscription period. Possible values: `'free_trial' | 'pay_as_you_go' | 'pay_up_front'` | | activePromotionalOfferId | string (Optional) | An id of an active promotional offer | | isSandbox | bool | True if the product was purchased in a sandbox environment |

AdaptyProfile.NonSubscription

Information about the user's non-subscription purchases.

| Name | Type | Description | |----|----|----------| | purchaseId | string | An identifier of the purchase in Adapty. You can use it to ensure that you've already processed this purchase (for example tracking one time products) | | vendorProductId | string | An identifier of the product in a store | | vendorTransactionId | string (optional) | Transaction id in a store | | store | string | A store of the purchase. Possible values: `'app_store' | 'play_store' | 'adapty'` | | purchasedAt |

iOS: Date

Android: string (ISO 8601 datetime)

| Date when the product was purchased | | isRefund | boolean | True if the purchase was refunded | | isConsumable | boolean | True if the product is consumable | | isSandbox | boolean | True if the product was purchased in a sandbox environment |

AdaptySubscriptionUpdateParameters

(*Android only*)
Parameters to change one subscription to another.

| Name | Type | Description | | :---------- | :----- | :---------- | | oldSubVendorProduct
string | The identifier of current subscription in Play Market that needs to be replaced. | | prorationMode | enum | Enum that corresponds to `BillingFlowParams.ProrationMode` values. |

---

title: "Handle errors" description: ""

metadataTitle: ""

Adapty returns errors for you to handle. We've categorized them by framework for your convenience:

- Handle errors in iOS
- Handle errors in Android
- Handle errors in Flutter, React Native, Unity

---

title: "iOS - Handle errors" description: "Learn how to effectively manage errors in iOS development with Adapty SDK's AdaptyError, featuring detailed properties for troubleshooting common issues."

metadataTitle: "iOS Error Handling: AdaptyError Overview"

Adapty SDK has its own wrapper for all kinds of errors, called `AdaptyError`. Basically, every error returned by an SDK is `AdaptyError`. It has two useful properties: `originalError` and `adaptyErrorCode`, described below.

originalError contains an original error in case you need the original one to work with. Can be SKError, NSError or just general Swift Error. This property is optional since some errors can be generated directly by SDK, like inconsistent or missing data, and won't have the original error around which the wrapper was initially built.

adaptyErrorCode can be used to handle common issues, like:

- invalid credentials
- network errors
- cancelled payments
- billing issues
- invalid receipt
- and much more

It's pretty easy to check the error for specific codes and react to them accordingly.

```
swift title="Swift" Adapty.makePurchase(product: product) { result in if result.error?.adaptyErrorCode == .paymentCancelled { // purchase was cancelled // you can offer discount to your user or remind them later } }
```

StoreKit errors

| Error | Code | Description | |-----|----|----------| |unknown | 0 | Error code indicating that an unknown or unexpected error occurred. | | clientInvalid | 1 | This error code indicates that the client is not allowed to perform the attempted action. | | paymentCancelled | 2 |

This error code indicates that the user canceled a payment request.

No action is required, but in terms of the business logic, you can offer a discount to your user or remind them later.

| | paymentInvalid | 3 | This error indicates that one of the payment parameters was not recognized by the App Store. | | paymentNotAllowed | 4 | This error code indicates that the user is not allowed to authorize payments. | | storeProductNotAvailable | 5 | This error code indicates that the requested product is not available in the store. | | cloudServicePermissionDenied | 6 | This error code indicates that the user has not allowed access to Cloud service information. | | cloudServiceNetworkConnectionFailed | 7 | This error code indicates that the device could not connect to the network. | | cloudServiceRevoked | 8 | This error code indicates that the user has revoked permission to use this cloud service. | | privacyAcknowledgementRequired | 9 | This error code indicates that the user has not yet acknowledged Appleâ€™s privacy policy. | | unauthorizedRequestData | 10 | This error code indicates that the app is attempting to use a property for which it does not have the required entitlement. | | invalidOfferIdentifier | 11 |

The offer `identifier` is not valid. For example, you have not set up an offer with that identifier in the App Store, or you have revoked the offer.

Make sure you set up desired offers in AppStore Connect and pass a valid offer identifier.

| | invalidSignature | 12 | This error code indicates that the signature in a payment discount is not valid. | | missingOfferParams | 13 | This error code indicates that parameters are missing in a payment discount. | | invalidOfferPrice | 14 | This error code indicates that the price you specified in App Store Connect is no longer valid. Offers must always represent a discounted price. | | noProductIDsFound | 1000 |

This error indicates that none of the products in the paywall is available in the store.

If you are encountering this error, please follow the steps below to resolve it:

1. Check if all the products have been added to the Adapty Dashboard.

2. Ensure that the Bundle ID of your app matches the one from Apple Connect.

3. Verify that the product identifiers from the app stores match the ones you have added to the Dashboard. Please note that the identifiers should not contain Bundle ID unless it is already included in the store.

4. Confirm that the app paid status is active in your Apple tax settings. Ensure that your tax information is up-to-date and your certificates are valid.

5. Check if a bank account is attached to the app, so it can be eligible for monetization.

6. Check if the products are available in all regions.

Also, ensure that your products are in **â€œReady to Submitâ€** state.

| | noProductsFound | 1001 | This error indicates that the product requested for purchase is not available in the store. | | productRequestFailed | 1002 | Unable to fetch available products at the moment. | | cantMakePayments | 1003 | In-app purchases are not allowed on this device. | | noPurchasesToRestore | 1004 | This error indicates that Google Play did not find the purchase to restore. | | cantReadReceipt | 1005 |

There is no valid receipt available on the device. This can be an issue during sandbox testing.

In the sandbox, you won't have a valid receipt file until you actually make a purchase, so make sure you do one before accessing it. During sandbox testing also make sure you signed in on a device with a valid Apple sandbox account.

| | productPurchaseFailed | 1006 | Product purchase failed. | | missingOfferSigningParams | 1007 |

This error indicates issues with Adapty integration or with offers.

Refer to the Configure App Store integration and to Offers for details on how to set them up.

|

Network errors

| Error | Code | Description | | :------------ | :--- | :---------- | | notActivated | 2002 | You need to properly configure Adapty SDK by `Adapty.activate` method. | | badRequest | 2003 | Bad request. | | serverError | 2004 | Server error. | | networkFailed | 2005 | The error indicates issues with the network connection. | | decodingFailed | 2006 | This error indicates that response decoding failed. | | encodingFailed | 2009 | This error indicates that request encoding failed. | | missingURL | 2010 | The requested URL is nil. |

General errors

| Error | Code | Description | | :------------------ | :--- | :---------- | | analyticsDisabled | 3000 | We can't handle analyt events, since you've opted it out. | | wrongParam | 3001 | This error indicates that some of your parameters are not correct do match the constraints. | | activateOnceError | 3005 | It is not possible to call `.activate` method more than once. | | profileWasChanged | 3006 | The user profile was changed during the operation. | | persistingDataError | 3100 | It was an error while saving data. | | operationInterrupted | 9000 | This operation was interrupted by the system. |

---

title: "Android - Handle errors" description: "Discover how to streamline error management in Android development with Adapty SDK's AdaptyError, providing detailed troubleshooting capabilities for comprehensive error handling."

metadataTitle: "Android Error Handling: AdaptyError Overview"

Every error is returned by the SDK is `AdaptyError`.

| Error | Code | Description | |-----|----|----------| | UNKNOWN | 0 | This error indicates that an unknown or unexpected error occurred. | |USER*CANCELED* | 2 | This error indicates that the user canceled a payment request. | | ITEMUNAVAILABLE | 5 | This error mostly happens at the testing stage. It may mean that the products are absent from production or that the user does not belong to the Testers group in Google Play. | | ADAPTY*NOT*INITIALIZED | 20 | You need to properly configure Adapty SDK by `Adapty.activate` method. | | PRODUCT*NOT*FOUND | 22 | This error indicates that the product requested for purchase is not available in the store. | | INVALID*JSON* | *23* |

The local fallback paywall JSON is not valid.

Fix your default English paywall, after that replace invalid local paywalls. Refer to the Customize paywall with remote config topic for details on how to fix a paywall and to the Define local fallback paywalls for details on how to replace the local paywalls.

| |

CURRENT

SUBSCRIPTION*TO*UPDATE

_NOT*FOUND*IN*HISTORY*

| 24 | The original subscription that needs to be replaced is not found in active subscriptions. | | PENDINGPURCHASE | 25 | This error indicates that the purchase state is pending rather than purchased. Refer to the Handling pending transactions page in the Android Developer docs for details. | | BILLING*SERVICE*TIMEOUT | 97 | This error indicates that the request has reached the maximum timeout before Google Play can respond. This could be caused, for example, by a delay in the execution of the action requested by the Play Billing Library call. | | FEATURE*NOT*SUPPORTED | 98 | The requested feature is not supported by the Play Store on the current device. | | BILLING*SERVICE*DISCONNECTED | 99 | This error indicates that the client appâ€™s connection to the Google Play Store service via the `BillingClient` has been severed. | | BILLING*SERVICE*UNAVAILABLE | 102 | This error indicates the Google Play Billing service is currently unavailable. In most cases, this means there is a network connection issue anywhere between the client device and Google Play Billing services. | | BILLING*UNAVAILABLE* | 103 |

This error indicates that a user billing error occurred during the purchase process. Examples of when this can occur include:

1. The Play Store app on the user's device is out of date.

2. The user is in an unsupported country.

3. The user is an enterprise user, and their enterprise admin has disabled users from making purchases.

4. Google Play is unable to charge the userâ€™s payment method. For example, the user's credit card might have expired.

| | DEVELOPERERROR | 105 | This error indicates you're improperly using an API. | | BILLING*ERROR* | 106 | This error indicates an internal problem with Google Play itself. | | ITEMALREADY*OWNED* | 107 | The product has already been purchased. | | ITEMNOT*OWNED* | 108 | This error indicates that the requested action on the item failed since it is not owned by the user. | | BILLINGNETWORK*ERROR* | 112 | This error indicates that there was a problem with the network connection between the device and Play systems. | | NOPRODUCT*IDS*FOUND | 1000 |

This error indicates that none of the products in the paywall is available in the store.

If you are encountering this error, please follow the steps below to resolve it:

1. Check if all the products have been added to Adapty Dashboard.

2. Ensure that the Bundle ID of your app matches the one from the Apple Connect.

3. Verify that the product identifiers from the app stores match with the ones you have added to the Dashboard. Please note that the identifiers should not contain Bundle ID, unless it is already included in the store.

4. Confirm that the app paid status is active in your Apple tax settings. Ensure that your tax information is up-to-date and your certificates are valid.

5. Check if a bank account is attached to the app, so it can be eligible for monetization.

6. Check if the products are available in all regions.Also, ensure that your products are in **â€œReady to Submitâ€** state.

| | NO*PURCHASES*TORE*RESTORE* | *1004* | *This error indicates that Google Play did not find the purchase to restore.* | | *AUTHENTICATION*ERROR | 2002 | You need to properly configure Adapty SDK by `Adapty.activate` method. | | BAD*REQUEST* | *2003* | *Bad request.* | | *SERVER*ERROR | 2004 | Server error. | | REQUEST*FAILED* | *2005* | *This error indicates a network issue that cannot be properly defined.* | | *DECODING*FAILED | 2006 | We could not decode the response. | | ANALYTICS*DISABLED* | *3000* | *We can't handle analytics events, since you've opted it out.* | | *WRONG*PARAMETER | 3001 | This error indicates that some of your parameters are not correct: blank when it cannot be blank or wrong type, etc. |

---

title: "Flutter, React Native, Unity - Handle errors" description: "Discover how to streamline error management in Fluter, React Native, and Unity development with Adapty SDK's AdaptyError, providing detailed troubleshooting capabilities for comprehensive error handling"

metadataTitle: "Fluter, React Native, and Unity Error Handling: AdaptyError Overview"

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Every error is returned by the SDK is `AdaptyErrorCode`. Here is an example:

```
javascript try { final result = await adapty.makePurchase(product: product); } on AdaptyError catch (adaptyError) { if (adaptyError.code == AdaptyErrorCode.paymentCancelled) { // Cancelled } } catch (e) { } csharp Adapty.MakePurchase(product, (profile, error) => { if (error != null && error.Code == Adapty.ErrorCode.PaymentCancelled) { // payment cancelled } });

typescript try { const params: MakePurchaseParamsInput = {}; await adapty.makePurchase(product, params); } catch (error) { if ( error instanceof AdaptyError && error.adaptyCode === getErrorCode(ErrorCode['2']) ) { // payment cancelled } }
```

Â System StoreKit codes

| Error | Code | Description | |-----|----|----------| | unknown | 0 | This error indicates that an unknown or unexpected error occurred. | | clientInvalid | 1 | This error code indicates that the client is not allowed to perform the attempted action. | | paymentCancelled | 2 |

This error code indicates that the user canceled a payment request.

No action is required, but in terms of the business logic, you can offer a discount to your user or remind them later.

| | paymentInvalid | 3 | This error indicates that one of the payment parameters was not recognized by the store. | | paymentNotAllowed | 4 |

This error code indicates that the user is not allowed to authorize payments. Possible reasons:

- Payments are not supported in the user's country.

- The user is a minor.

| | storeProductNotAvailable | 5 | This error code indicates that the requested product is absent from the App Store. Make sure the product is available for the used country. | | cloudServicePermissionDenied | 6 | This error code indicates that the user has not allowed access to Cloud service information. | | cloudServiceNetworkConnectionFailed | 7 | This error code indicates that the device could not connect to the network. | | cloudServiceRevoked | 8 | This error code indicates that the user has revoked permission to use this cloud service. | | privacyAcknowledgementRequired | 9 | This error code indicates that the user has not yet acknowledged the store privacy policy. | | unauthorizedRequestData | 10 | This error code indicates that the request is built incorrectly. | | invalidOfferIdentifier | 11 |

The offer identifier is not valid. Possible reasons:

- You have not set up an offer with that identifier in the App Store.

- You have revoked the offer.

- You misprinted the offer ID.

| | invalidSignature | 12 | This error code indicates that the signature in a payment discount is not valid. Make sure you've filled out the **In-app purchase Key ID** field and uploaded the **In-App Purchase Private Key** file. Refer to the Configure App Store integration topic for details. | | missingOfferParams | 13 |

This error indicates issues with Adapty integration or with offers.

Refer to the Configure App Store integration and to Offers for details on how to set them up.

| | invalidOfferPrice | 14 | This error code indicates that the price you specified in the store is no longer valid. Offers must always represent a discounted price. |

Custom Android codes

| Error | Code | Description | |-----|----|----------| | adaptyNotInitialized | 20 | You need to properly configure Adapty SDK by`Adapty.activate` method. Learn how to do it for Flutter, for React Native, and for Unity. | | productNotFound | 22 | This error indicates that the product requested for purchase is not available in the store. | | invalidJson | 23 | The paywall JSON is not valid. Fix it in the Adapty Dashboard. Refer to the Customize paywall with remote config topic for details on how to fix it. | | currentSubscriptionToUpdateNotFoundInHistory | 24 | The original subscription that needs to be renewed is not found. | | pendingPurchase | 25 | This error indicates that the purchase state is pending rather than purchased. Refer to the Handling pending transactions page in the Android Developer docs for details. | | billingServiceTimeout | 97 | This error indicates that the request has reached the maximum timeout before Google Play can respond. This could be caused, for example, by a delay in the execution of the action requested by the Play Billing Library call. | | featureNotSupported | 98 | The requested feature is not supported by the Play Store on the current device. | | billingServiceDisconnected | 99 | This fatal error indicates that the client appâ€™s connection to the Google Play Store service via the `BillingClient` has been severed. | | billingServiceUnavailable | 102 | This transient error indicates the Google Play Billing service is currently unavailable. In most cases, this means there is a network connection issue anywhere between the client device and Google Play Billing services. | | billingUnavailable | 103 |

This error indicates that a user billing error occurred during the purchase process. Examples of when this can occur include:

1. The Play Store app on the user's device is out of date.

2. The user is in an unsupported country.

3. The user is an enterprise user, and their enterprise admin has disabled users from making purchases.

4. Google Play is unable to charge the userâ€™s payment method. For example, the user's credit card might have expired.

| | developerError | 105 | This is a fatal error that indicates you're improperly using an API. | | billingError | 106 | This is a fatal error that indicates an internal problem with Google Play itself. | | itemAlreadyOwned | 107 |

The consumable product has already been purchased. | | itemNotOwned | 108 | This error indicates that the requested action on the item failed sin |

Custom StoreKit codes

| Error | Code | Description | |-----|----|----------| | noProductIDsFound | 1000 |

This error indicates that none of the products in the paywall is available in the store.

If you are encountering this error, please follow the steps below to resolve it:

1. Check if all the products have been added to Adapty Dashboard.

2. Ensure that the Bundle ID of your app matches the one from the Apple Connect.

3. Verify that the product identifiers from the app stores match with the ones you have added to the Dashboard. Please note that the identifiers should not contain Bundle ID, unless it is already included in the store.

4. Confirm that the app paid status is active in your Apple tax settings. Ensure that your tax information is up-to-date and your certificates are valid.

5. Check if a bank account is attached to the app, so it can be eligible for monetization.

6. Check if the products are available in all regions.Also, ensure that your products are in **â€œReady to Submitâ€** state.

| | productRequestFailed | 1002 |

Unable to fetch available products at the moment. Possible reason:

- No cache was yet created and no internet connection at the same time.

| | cantMakePayments | 1003 | In-App purchases are not allowed on this device. | | noPurchasesToRestore | 1004 | This error indicates that Google Play did not find the purchase to restore. | | cantReadReceipt | 1005 |

There is no valid receipt available on the device. This can be an issue during sandbox testing.

No action is required, but in terms of the business logic, you can offer a discount to your user or remind them later.

| | productPurchaseFailed | 1006 | Product purchase failed. | | refreshReceiptFailed | 1010 | This error indicates that the receipt was not received. Applicable to StoreKit 1 only. | | receiveRestoredTransactionsFailed | 1011 | Purchase restoration failed. |

Custom network codes

| Error | Code | Description | | :---------- | :------ | :---------- | | notActivated | 2002 | You need to properly configure Adapty SDK by `Adapty.activate` method. Learn how to do it for Flutter , for React Native , and for Unity . | | badRequest | 2003 | Bad request. | | serverError | 2004 | Server error. | | networkFailed | 2005 | The network request failed. | | decodingFailed | 2006 | This error indicates that response decoding failed. | | encodingFailed | 2009 | This error indicates that request encoding failed. | | analyticsDisabled | 3000??? | We can't handle analytics events, since you've opted it out. Refer to the Analytics integration topic for details. | | wrongParam | 3001 | This error indicates that some of your parameters are not correct: blank when it cannot be blank or wrong type, etc. | | activateOnceError | 3005 | It is not possible to call `.activate` method more than once. | | profileWasChanged | 3006 | The user profile was changed during the operation. | | fetchTimeoutError | 3101 | This error means that the paywall could not be fetched within the set limit. To avoid this situation, set up local fallbacks. | | operationInterrupted | 9000 | This operation was interrupted by the system. |

---

title: "Test purchases" description: ""

metadataTitle: ""

After you have set up everything in the Adapty Dashboard and your mobile app, it's time to test the in-app purchases.

There is a list of issues that you may encounter upon testing purchases in Sandbox. It is both for the purchase flow itself, and the results youâ€™ll see in the app/on your dashboard.

- iOS
- Android

---

title: "Test in-app purchases in Apple App Store" description: ""

metadataTitle: ""

Once you've configured everything in the Adapty Dashboard and your mobile app, it's time to conduct in-app purchase testing.

:::warning Test on a real device

Whatever tool you choose, it's essential to conduct testing on a real device to validate the end-to-end purchase process. While testing on a simulator allows you to examine paywalls, it does not enable interaction with the Apple's servers, making it impossible to test purchases. :::

Note: none of the test tools charge users when they test buying a product. The App Store doesnâ€™t send emails for purchases or refunds made in the test environments.

You can either choose to test in Sandbox or using TestFlight. Sandbox is the best choice when you as a developer want to test the purchases yourself on a device linked to your Mac with XCode, while TestFlight is more convenient for other members of the team.

Choose the method that works best for you:

- Testing in Sandbox
- Testing via TestFlight

---

title: "Test in-app purchases in App Store Sandbox" description: "Learn how to prepare for purchase testing using App Store Sandbox, ensuring smooth testing processes on real devices with Sandbox Apple IDs. Follow this guide to seamlessly validate your app's purchase flow"

metadataTitle: "Testing Purchases in App Store Sandbox: A Step-by-Step Guide"

Once you've configured in-app purchases in your mobile app, it's crucial to test them thoroughly to ensure functionality and proper transmission of transactions to Adapty before releasing the app to production. Transactions and purchases that occur in the sandbox donâ€™t incur charges. To conduct sandbox testing, you'll need to use a special test account - Sandbox Apple ID, and ensure the testing device is added to the Developer Account in the App Store Connect.

Sandbox testing is ideal for developers who wish to personally test purchases on a device connected to their Mac via XCode.
For more details, you can refer to the Apple's documentation on Testing in-app purchases with sandbox.

:::warning Test on a real device

To validate the end-to-end purchase process, it's essential to conduct testing on a real device. While testing on a simulator allows you to examine paywalls, it does not enable interaction with the Apple server, making it impossible to test purchases. :::

Before you start testing

Before you start testing in-app purchases, make sure that:

1. Your Apple Developer Program account is active. For more information, see Apple's What you need to enroll.
2. Your membership Account Holder has signed the Paid Applications Agreement, as described in Apple's Sign and update agreements.
3. You set up the product information in App Store Connect for the app youâ€™re testing. At a minimum, set up a product reference name, product ID, a localized name, and a price.
4. The **Keychain Sharing** capability is disabled. For more information, see Apple's article Configuring keychain sharing.
5. Youâ€™re running a development-signed rather than a production-signed build of your app.
6. You have completed all the steps outlined in the release checklist.

Prepare for Sandbox testing

Testing in-app purchases in the sandbox environment doesnâ€™t involve uploading your app binary to App Store Connect. Instead, you build and run your app directly from Xcode. However, it does require a special test account - Sandbox Apple ID.

Step 1. Create a Sandbox test account (Sandbox Apple ID) in the App Store Connect

:::warning Create a new sandbox test account

When testing your purchases, it's crucial to create a new sandbox test account each time. This ensures a clean purchase history, optimal performance, and smooth functionality. :::

To create a Sandbox Apple ID:

1. Open **App Store Connect**. Proceed to **Users and Access â†' Sandbox â†' Test Accounts** section.

2. Click the add button **(+)** button next to the **Test Accounts** title.

3. In the **New Tester** window, enter the data of the test user.

   :::warning

   - Make sure to provide a valid email you can verify.
   - Make sure to define the **Country or Region** which you plan to test. :::
4. Click the **Create** button to confirm the creation

Step 3. Add the Sandbox test account to your device

The first time you run an app from XCode on your device, there's no need to manually add a Sandbox account. Upon building the app in XCode and running it on your device, when you initiate a purchase, the device prompts you to enter the Apple ID for the purchase. Simply enter your Sandbox Apple ID and password at this juncture, and the Sandbox test account will be automatically added to your device.

If you need to change the Sandbox Apple ID associated with your device, you can do so directly on the device by following these steps:

1. On iOS 12, navigate to **Settings > [Your Account] > App Store > Sandbox Account**.
   On iOS 13 or greater, navigate to **Settings > App Store > Sandbox Account**.
2. Tap the current Sandbox Apple ID in the **Sandbox Account** section.
3. Tap the **Sign Out** button.
4. Tap the **Sign In** button.
5. In the **Use the Apple IS for Apple Media Services** window, tap the **Use Other Apple ID** button.
6. In the **Apple ID Sign-In Requested** window, enter the new sandbox account credentials that you previously created.
7. Tap the **Done** button.
8. In the **Apple ID Security** window, tap the **Other options** button.
9. In the **Protect your account** window, tap the **Do not upgrade** button.

The added sandbox account is shown in the **Sandbox Account** section of your iOS device **Settings**.

Step 4. Connect the device to your Mac with XCode

To execute the built app version on your real device, include the device as a run destination in the Xcode project

1. Connect your real device to the Mac with XCode using a cable or using the same Wi-Fi.
2. Choose the **Windows** -> **Devices and Simulators** from the XCode main menu.
3. In the **Devices** tab, choose your device.
4. Tap the **Trust** button on your mobile phone.

Your device is connected to the XCode and can be used for sandbox testing.

Step 5. Build the app and run it

Click the **Run** button in the toolbar or choose **Product -> Run** to build and run the app on the connected real device. If the build is successful, Xcode runs the app on your iOS device and opens a debugging session in the debug area of the XCode.

The app is ready for testing on the device.

:::note When youâ€™re done testing the app, click the **Stop** button in the XCode toolbar. :::

Step 6. Make purchase

Make a purchase in your mobile app via paywall.

:::info Now you can validate that the test purchase is successful. :::

---

title: "Test in-app purchases with TestFlight" description: "Learn how to prepare for purchase testing using TestFlight, ensuring smooth testing processes on real devices with genuine Apple accounts. Follow this guide to seamlessly validate your app's purchase flow"

metadataTitle: "Testing Purchases with TestFlight: A Step-by-Step Guide"

TestFlight lets you get feedback from members of your team. Testing is done on real devices, and the testers will need to use their real Apple accounts. Please note that TestFlight uses the sandbox environment for in-app purchases. Transactions and purchases donâ€™t incur charges in this case.
Testing with TestFlight on a sandbox environment is the best choice when you want to let your team members test your app. For more details, you can refer to Apple's documentation on Beta testing with TestFlight.

:::warning Test on a real device

To validate the end-to-end purchase process, it's essential to conduct testing on a real device. While testing on a simulator allows you to examine paywalls, it does not enable interaction with the Apple server, making it impossible to test purchases. :::

Before you start testing

Before you start testing in-app purchases, make sure that:

1. Your Apple Developer Program account is active. For more information, see Apple's What you need to enroll.
2. Your membership Account Holder has signed the Paid Applications Agreement, as described in Apple's Sign and update agreements.
3. You set up the product information in App Store Connect for the app youâ€™re testing. At a minimum, set up a product reference name, product ID, a localized name, and a price.
4. The **Keychain Sharing** capability is disabled. For more information, see Apple's article Configuring keychain sharing.
5. Youâ€™re running a development-signed rather than a production-signed build of your app.
6. You have completed all the steps outlined in the release checklist.

Prepare for testing with TestFlight

When conducting purchase testing with TestFlight, ensure you're using a real device and your genuine Apple account. Notably, transactions and purchases made during testing won't result in any charges when using the development-signed build of your app.

To test beta versions of apps using TestFlight:

1. Build your mobile app version and send it to TestFlight without releasing it via App Store Connect.
2. On your iOS device, install the TestFlight.
3. Share the link to your app built into the device and tap it on your device.
4. If youâ€™re a new tester for the app, tap **Accept**.
5. Tap **Install** to install the app on your device.

The app is installed and ready for testing.

Make purchase

Make a purchase in your mobile app via paywall.

:::info Now you can validate that the test purchase is successful. :::

---

title: "Test in-app purchases in Google Play Store" description: ""

metadataTitle: ""

Testing in-app purchases (IAPs) in your Android app can be a crucial step before releasing your app to the public. Sandbox testing is a safe and efficient way to test IAPs without charging real money to your users. In this guide, we'll walk you through the process of sandbox testing IAPs on the Google Play Store for Android.

Test your app on a real device

To ensure optimal performance of your Android app, it's recommended that you test it on a real device instead of an emulator. While we have successfully tested on emulators, Google recommends using a real device.

If you do decide to use an emulator, make sure that it has Google Play installed. This will help ensure that your app is functioning properly.

Create a test user for app testing

To facilitate testing during later stages of development, you'll need to create a test user. This user will be the first account you log in with on your Android testing device.

Note that the primary account on an Android device can only be changed by performing a factory reset. Therefore, it's important to create a separate test user account to avoid having to perform a factory reset on your device.

Configure licence testing for your app

Once you've created a test user account, you'll need to configure licensing testing for your app. To do this, follow these steps:

1. In the Console sidebar, navigate to **Setup**.
2. Select **License testing**.
3. Add the account that you're using on your real device (i.e., the account you're currently logged in with) to the list.

This will allow you to configure licensing testing for your app and ensure that it's functioning properly.

In our example, we already have a list of testers:

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Create a closed track and add the test user to it

1. Publish a signed version of your app to a closed track. If you haven't created a closed track yet, you can create one in the **Closed testing** section of the **Testing** menu.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Just as previously, you can use one of the existing lists or create a new one:

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Press **Enter**, and click the **Save changes** button.

3. Open the **Opt-in URL** in your testing device to make the user a tester. You can send the URL to your device via email, for example.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::warning Important

Opening the opt-in URL marks your Play account for testing. If you don't complete this step, products will not load. :::

:::warning Check Your Application ID

Often developers will use a different application ID for their test builds. This will cause you problems since Google Play Services uses the application ID to find your in-app purchases. :::

:::warning Add a PIN to the test device if needed

There are cases where a test user may be allowed to purchase consumables, but not subscriptions, if the test device does not have a PIN. This may manifest in a cryptic "Something went wrong" message. Make sure that the test device has a PIN, and that the device is logged into Google Play Store. :::

Upload a signed APK to the closed track

Generate a signed APK or use Android App Bundle to upload a signed APK to the closed track you just created. You don't even need to roll out the release. Just upload the APK. You can find more information about this in this support article.

> ðŸŒŽ Make your release available in at least one country
>
> If your app is new, you may need to make it available in your country or region. To do so, go to Testing > Closed testing, click on your test track, and go to Countries/regions to add the desired countries and regions.

Test in-app purchases

After you've uploaded the APK, wait a few minutes for the release to process. Then, open your testing device and sign in with the email account you added to the Testers list. You can then test in-app purchases as you would on a production app.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

If you run into any issues, refer to the documentation or contact Google Play Developer support.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Validate test purchases" description: "Verify the success of test purchases by checking their tracking in Adapty's Event Feed. Learn how to ensure each transaction is accurately recorded for seamless testing of your mobile app's purchase flow"

metadataTitle: "Validating Test Purchases: Ensure Success with Adapty"

Before releasing your mobile app to production, it's crucial to test in-app purchases thoroughly. Please refer to ourTest in-app purchases in Apple App Store and Test in-app purchases in Google Play Store topics for detailed guidance on testing. Once you begin testing, you need to verify the success of test purchases.

Every time you make a test purchase on your mobile device, view the corresponding transaction in the **Event Feed** in the Adapty Dashboard. If the purchase does not appear in the **Event Feed**, it's not being tracked by Adapty.

âœ… Test purchase is successful

If the test purchase is successful, its transaction event will be dsplayed in the **Event Feed**:

feedsandbox.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

If transactions work as expected, proceed to the Release checklist, and then proceed with the app release.

âŒ Test purchase is not successful

If you observe no transaction event within 10 minutes or encounter an error in the mobile app, refer to the Troubleshooting and articles on error handling for iOS, for Android, for Flutter, React Native, and Unity for

potential solutions.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto' /* center alignment */ }} />

---

title: "Troubleshooting test purchases" description: "Encounter transaction issues during test purchases with Adapty? Follow our tailored troubleshooting guide to resolve errors, pricing discrepancies, and transaction discrepancies, ensuring seamless testing of your mobile app's purchase flow"

metadataTitle: "Troubleshooting Test Purchases with Adapty: Solutions for Transaction Issues"

If you encounter transaction issues, please first make sure you have completed all the steps outlined in the release checklist. If you've completed all the steps and still encounter issues, please follow the guidance provided below to resolve them:

| Issue | Solution | |-----|--------| | An error is returned in the mobile app | Refer to the error list for your platform: for iOS, for Android, for Flutter, React Native, or Unity and follow our recommendations to resolve the issue. | | Transaction is absent from the **Event Feed** although no error is returned in the mobile app |

1. for iOS: Ensure you use a real device rather than a simulator.

2. Ensure that the `Bundle ID/Package name` of your app matches the one in the **App settings**.

3. Ensure the `PUBLIC_SDK_KEY` in your app matches the **Public SDK key** in the Adapty Dashboard: **App settings-> General** tab -> **API keys** subsection.

| | No event is present in my testing profile |

A new user profile record is automatically created in Adapty in the following cases:

- When a user runs your app for the first time

- When a user logs out of your app

All transactions and events in a chain are tied to the profile that generated the first transaction. This helps keep the entire transaction history, including trial periods, subscription purchases, renewals, and more linked to the same profile. This means that a new profile record that generated a subsequent translation - we call it a non-original profile - may not have any events associated with it but will retain the granted access level. In some cases, you'll also see "access*level*updated" events here. It's advised to ignore these new, empty profiles and focus on the original ones for a clearer view of the user's transaction history.

This behavior is more prominent for testing scenarios but also normal for production. It should not be treated as a bug or SDK misconfiguration.In order to run the test with only one profile and have fun concise history in one place, create a new test account (Sandbox Apple ID) each time you reinstall the mobile app.

Please see an example of a non-original profile under the table. For more details on profile creation, please see Profile record creation.

| | Prices do not reflect the actual prices set in App Store Connect |

In both Sandbox and TestFlight which uses the sandbox environment for in-app purchases, it's important to verify that the purchase flow functions correctly, rather than focusing on the accuracy of prices. It's worth noting that Apple's API can occasionally provide inaccurate data, particularly when different regions are configured for devices or accounts. And since the prices come directly from the Store and the Adapty backend does not affect purchase prices in any way, you can ignore any inaccuracy in prices during the testing of the purchases through Adapty.

Therefore, prioritize testing the purchase flow itself over the accuracy of prices to ensure it functions as intended.

| | The transaction time in the **Event Feed** is incorrect | The **Event Feed** uses the time zone set in the **App Settings**. To align the time zone of events with your local time, adjust the **Reporting timezone** in **App settings -> General** tab. |

Here is an example of a non-original profile. Notice the absence of events in the **User history** and the presence of an access level.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto' /* center alignment */ }} />

---

title: "Testing devices" description: "Discover how to assign your device as a test device in Adapty to bypass caching and view immediate changes to paywalls and placements"

metadataTitle: "How to Mark Devices as Test in Adapty for Immediate Changes"

Due to caching, changes made to paywalls or placements may take up to 20 minutes to reflect on the device. This delay is needed to speed up fetching a paywall for regular users, but it can be inconvenient when testing your changes.

For testing purposes, you can assign your device as test, which will disable caching and ensure that your changes are immediately displayed.

:::note Testing devices are supported starting from specific SDK versions:

- iOS: 2.11.1
- Android: 2.11.3
- React Native: 2.11.1

Flutter and Unity support will be added later. :::

Mark your device as test

1. Open the **App settings** in the Adapty Dashboard.
2. Scroll down to the **Test devices** section in the **General** tab.

deviceadd.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto' /* center alignment */ }} />

3. Click the **Add test device** button.

usersadd_device.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto' /* center alignment */ }} />

4. In the **Add test device** window, enter:

   | Field | Description | | :---------- | :---------- | | **Test device name** | Name of the test device(s) for your reference. | | **ID used to identify this test device** | Choose the identifier type you plan to use to identify the test device(s). Follow our recommendations in the Which identifier you should use section below to pick the best option. | | **ID value** | Enter the value of the identifier. |

5. Remember to click **Add test device** button to save the changes.

Which identifier you should use

To identify a device, you can use several identifiers. We recommend the following:

- **Customer User ID** for both iOS and Android devices if you identify your users in Adapty. That is the best choice, especially if you have more that one test device for one account in your app. If Customer User ID is used as **ID used to identify this test device**, all the devices connected to this account will be marked as test devices.
- **IDFA (iOS)** and **Advertising ID (Android)**: These advertising identifiers are a perfect choice for iOS and Android devices respectively if you're already asking your users for consent to access them. Even if you have a Customer User ID, you may prefer using advertising identifiers if you switch between accounts in your app while testing. Additionally, those identifiers are beneficial if the same account has both test and personal devices and you don't want the personal devices marked as test devices.

There are other options, such as the Adapty Profile ID, IDFV, and Android ID, which are less convenient but can be used if you cannot use Customer User ID, IDFA, or Advertising ID.

Let's review all possible options in detail.

Identifiers for all platforms

| Identifier | Usage | |----------|-----| | Customer User ID |

A unique identifier set by you to identify your users in your system. This could be the user's email, your internal ID, or any other string. To use this option, you must Identify your users in Adapty.

It is the best choice for identifying a test device, especially if you're using several devices for the same account. All the devices with this account will be considered test.

| | Adapty profile ID |

A unique identifier for the user profile in Adapty.

Use it if you cannot use Customer User ID, IDFA for iOS, or Advertising ID for Android. Note that the Adapty Profile ID can change if you reinstall the app or re-log in.

|

How to obtain Customer User ID and Adapty profile ID

Both identifiers can be obtained in the **Profile** details of the Adapty Dashboard:

1. Find the user's profile in the **Adapty Profiles -> Event feed** tab. :::note To find the exact profile, make a rare type of transaction. In this case, once the transaction appears in the **Event Feed**, you'll easily identify it. :::
2. Copy **Customer user ID** and **Adapty ID** field values in the profile details:

usersCUID*adapty*ID.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

Apple identifiers

| Identifier | Usage | |----------|-----| | IDFA |

The Identifier for Advertisers (IDFA) is a unique device identifier assigned by Apple to a userâ€™s device.

It's ideal for iOS devices as it never changes on its own, although you can manually reset it.

Note: Since the rollout of iOS 14.5, advertisers must ask for user consent to access the IDFA. Ensure you are asking for consent in your app and you have provided it on your test device.

| | IDFV | The Identifier for Vendors (IDFV) is a unique alphanumeric identifier assigned by Apple to all apps on a single device from the same publisher/vendor. It can change if you reinstall or update your app. |

How to obtain the IDFA

Apple does not provide the IDFA by default. Obtain it from the profile attribution in the Adapty Dashboard:

1. Find the user's profile in the **Adapty Profiles -> Event feed** tab. :::note To find the exact profile, make a rare type of transaction. In this case, once the transaction appears in the **Event Feed**, you'll easily identify it. :::
2. Open the profile details and copy the **IDFA** field value in the **Attributes** section:

usersidfa.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

Alternatively, you can find the app on the App Store that will show your IDFA to you.

How to obtain the Identifier for vendors (IDFV)

To obtain the IDFV, ask your developer to request it using the following method for your app and display the received identifier to your logs or debug panel.

```
swift title="Swift" UIDevice.current.identifierForVendor
```

Google identifiers

| Identifier | Usage | |----------|-----| | Advertising ID |

The Advertising ID is a unique device identifier assigned by Google to a userâ€™s device.

It's ideal for Android devices as it never changes on its own, although you can manually reset it.

Note: To use it, turn off the **Opt out of Ads Personalization** in your **Ads** settings if you use Android 12 or higher.

| | Android ID | The Android ID is a unique identifier for each combination of app-signing key, user, and device. Available on Android 8.0 and higher versions. |

How to obtain Advertising ID

To find your device's advertising ID:

1. Open the **Settings** app on your Android device.
2. Click on **Google**.
3. Select **Ads** under **Services**. Your advertising ID will be listed at the bottom of the screen.

How to obtain Android ID

To obtain the Android ID, ask your developer to request the ANDROID_ID using the following method for your app and display the received identifier in your logs or debug panel.

```
kotlin title="Kotlin/Java" android.provider.Settings.Secure.getString(contentResolver, android.provider.Settings.Secure.ANDROID_ID);
```

---

title: "Release checklist" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Weâ€™re thrilled youâ€™ve decided to use Adapty! We want you to get the best results from the very first build. This guide will walk you through how to get started with Adapty

Installing Adapty SDK

Install Adapty SDK in your app and be sure you have replaced the **"PUBLIC*SDK*KEY"** placeholder with your actual **Public SDK key**.

Bear in mind, that SDK calls must be made after calling* `.activate()`* method. Otherwise, we won't be able to authenticate requests and they will be canceled.

```
swift Adapty.activate("PUBLIC_SDK_KEY", customerUserId: "YOUR_USER_ID") kotlin override fun onCreate() { super.onCreate() Adapty.activate(applicationContext, "PUBLIC_SDK_KEY",
customerUserId: "YOUR_USER_ID") } xml <dict> ... <key>AdaptyPublicSdkKey</key> <string>PUBLIC_SDK_KEY</string> </dict> xml <application ...> ... <meta-data
android:name="AdaptyPublicSdkKey" android:value="PUBLIC_SDK_KEY" /> </application> ```typescript title="React Native - /src/App.tsx" import { activateAdapty } from 'react-native-adapty';
```

const App: React.FC = () => { // ... useEffect(() => { activateAdapty({ sdkKey: 'PUBLIC*SDK*KEY' }); }, []); // ... } ```

Configuring processing of purchases

Adding **App Store shared secret** for iOS and both **package name** with **service account key file** for Android would be neccessary to allow Adapty to successfully process purchasing events.

Subscription Events

Here is what you can do to set up tracking of subscription events.

| | | | :------------- | :---------- | | **For iOS** | Update the App Store Server Notifications with our link | | **For Android** | Set up Real-time Developer Notificatio (RTDN) |

Integrations

Integrations with third-party analytics and attribution services require passing identifiers to the SDK.

| | | | :---------- | :---------- | | **.updateProfile()** | Use this to passing identifiers to Amplitude, Mixpanel, Facebook Ads, and AppMetrica | | **.updateAttribution()** | This method would be required for passing attribution data from AppsFlyer, Adjust, and Branch. Be sure to configure the integration of interest in Adapty Dashboard, by providing API key and event names |

Promo campaigns and promo offers

If you want to use Adapty along with Apple Promotional Offers, adding a subscription key will allow us to sign offers.

Notes

:::warning Don't forget about Privacy Labels

Learn more about the data Adapty collects and which flags you'd need to set for a review. :::

:::danger Make sure to send paywall views to Adapty using **.logShowPaywall()** method. Otherwise, paywall views will not be accounted for in the metrics and conversions will be irrelevant. :::

If you have any questions about integrating Adapty SDK, feel free to contact us using the website (we use Intercom in the bottom right corner) or just email us at support@adapty.io.

---

title: "How Adapty analytics works" description: ""

metadataTitle: ""

Adapty Analytics offers a powerful suite of tools that provide valuable insights into your user base, allowing you to make data-driven decisions and optimize your app's performance. With Adapty, you can go beyond basic metrics and dive deep into advanced analytics such as funnels, cohorts, retention, lifetime value (LTV) charts, and more. Let's explore the general approach to data gathering, calculation, and the various analytics features available.

To learn more about specific metrics and advanced analytics features, please refer to the relevant sections in the documentation menu.

To get started with Adapty analytics, simply install the Adapty SDK for your iOS or Android apps. From there, you can analyze close to real-time metrics with advanced controls, such as filters and grouping, including country, paywall, product, and more. By utilizing this powerful tool, you can gain a deeper understanding of your user base and make data-driven decisions to optimize your app's performance.

Close to real-time data for advanced analysis

When it comes to analytics, the Apple App Store and Google Play Store offer some basic metrics such as downloads, revenue, and retention rates. More than that, the data is usually updated only once a day, which can limit your ability to make real-time decisions.

In contrast, Adapty offers close to real-time analytics that allows you to track key metrics as they happen, giving you an accurate and up-to-date view of your app's performance. Data coming to Adapty gets processed and ends up in analytics 15 to 30 minutes after the event has been received. This extra-processing step is necessary to make sure our analytics is fast and reliable.

With Adapty, you can access advanced metrics and filters that provide deeper insights into user behavior, such as ad network, ad campaign, country, paywall, product, and more. The analytics data is generated using the current state of purchase receipts stored in Adapty, so they always reflect up-to-date information without relying on any client-side event logging. This means you can access real-time metrics without any delay, ensuring that you always have the latest information on hand.

Additionally, Adapty allows you to attribute analytics data to specific customers, including data from Apple Search Ads and mobile measurement partners (Appsflyer, Adjust, Branch, etc). This gives you a complete understanding of your users and enables you to personalize your app's marketing and engagement strategies.

Data attribution

One of the advantages of Adapty analytics is that all data is attributed to every device, which allows for more precise segmentation based on device type, location, and other factors. This provides deeper insights into user behavior and enables you to tailor your marketing and engagement strategies to specific audiences.

Additionally, Adapty offers robust CRM functionality that allows you to track and analyze user behavior across multiple touchpoints, including in-app purchases, and subscriptions. This provides a complete view of the customer journey and helps you identify opportunities to optimize user acquisition, retention, and monetization. Learn more about the CRM in the Adapty system.

Data from both stores

In addition to providing a unified view of data from both the App Store and Play Store, Adapty analytics also enables you to group and filter data by store. This allows you to compare and contrast metrics between the two stores and gain deeper insights into user behavior and preferences.

For example, you can analyze how different user segments behave in each store, such as comparing retention rates for iOS with Android users or analyzing revenue generated by users in different countries in each store. You can also filter by different app versions, pricing tiers, and more, to gain granular insights into user behavior and make informed decisions.

Adapty events and integrations

In addition to providing store-specific metrics, Adapty also offers a powerful set of subscription events that can be used to gain deeper insights into user behavior. With 17 subscription events that can be sent to 3rd party analytics and webhook, you can gain granular insights into user behavior and optimize your subscription strategies for maximum impact. Check our documentation to learn more about Adapty events and 3rd party integrations.

Historical data import

Another advantage of Adapty's analytics platform is the ability to import historical data, giving you a more comprehensive view of your app's performance over time. This feature allows you to see transaction data from before you started using Adapty and organize it into cohorts for more granular analysis.

Adapty's analytics platform also offers advanced analytics and integrations with a range of third-party tools, allowing you to gain deeper insights into user behavior and optimize your subscription offerings accordingly. With Adapty, you can easily track key metrics such as conversion rates, churn, and lifetime value, as well as gain insights into user segments and cohorts.

Commission fee and taxes calculation

Adapty Analytics provides accurate calculation of commission fees and taxes associated with transactions.

Commission fee calculation: Adapty supports commission calculation for both the App Store and Play Store. The calculation is based on the Gross revenue, representing the total revenue generated by transactions before any deductions. For detailed information on how Adapty calculates store commission fees, please refer to the corresponding documentation for App Store and Play Store.

Tax calculation: Adapty supports tax calculation for both the App Store and Play Store transactions. We incorporate tax calculation into its revenue analytics. The platform considers the user's store account country VAT tax rate for the transaction to accurately calculate the taxes associated with each transaction. The taxes are deducted from the revenue after the commission fee is applied.

Predictions

Adapty's analytics platform offers predictive capabilities specifically for cohort analyses, focusing on Lifetime Value (LTV) and revenue predictions. By leveraging historical data and advanced machine learning algorithms, Adapty can generate predictions for these key metrics, providing valuable insights for planning and optimizing your app's growth strategies. To learn more about utilizing predictions for cohort analyses and understanding how to interpret and apply the forecasted insights, refer to the detailed documentation on Predictions in cohorts.

By leveraging Adapty's advanced analytics capabilities, you can make data-driven decisions and optimize your app's performance for success.

---

title: "Analytics overview" description: ""

metadataTitle: ""

Overview is a section of Adapty Dashboard that allows you to see multiple metrics in a single place. You can customize which charts you would like to see and view data for all of your apps at once (unlike inCharts which you can use to dive deeper into a particular app).

It's located right under Home in the left-side menu:

2024-01-22at_18.38.162x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Charts

Overview has the following charts available (you can click on the name to learn more about how we calculate it):

- Revenue
- MRR
- ARR
- ARPPU
- New trials
- New subscriptions
- Active trials
- Active subscriptions
- New non-subscriptions
- Refund events
- Refund money

You can customize which charts to show as well as their order. To do that, press Edit in the top-right corner and then either remove charts you don't need, add more or rearrange existing ones by drag and dropping. You can also customize Overview contents in the "Add" menu:

2024-01-22at_20.01.05.gif').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Controls

Controls for diving deeper into your data in Overview are very similar to what we have in Charts â€" and most of them are described in Analytics controls.

There is one important difference though: you can group and filter by country, store and, most notably, by app â€" as Overview shows data for all of your apps at once by default. This can be helpful to understand how each of your app contributes to your business metrics:

2024-01-22at_19.56.302x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note Timezone control

By default, the timezone for Overview is inherited from one of your apps. But if your apps have different reporting timezones in their settings, you can customise the timezone for Overview by pressing Edit and selecting the appropriate one in the dropdown that will appear. :::

---

title: "Analytics controls" description: ""

metadataTitle: ""

Adapty offers a wide range of controls to help you gain valuable insights and unlock the full potential of your data and gain a comprehensive view of your business performance. Whether you're analyzing charts, cohorts, funnels, retention, conversion data, or LTV, these controls provide powerful functionality. By leveraging these controls, you can dive deeper into your data and extract meaningful insights to drive your business decisions.

In the following article, you can learn more about each control and how to use them effectively. Additionally, you'll find information about which controls are supported for each type of analytics, including charts, cohorts, funnels, retention, and conversion.

| Control | Charts | Cohorts | Funnels | Retention | Conversion | LTV | | :-------------- | :----- | :------ | :------ | :-------- | :--------- | :-- | | Time ranges | âœ… | âœ… | âœ… | âœ… | âœ… | âœ… | | Data comparison | âœ… | âŒ | âŒ | âŒ | âŒ | | | Filtering | âœ… | âœ… | âœ… | âœ… | âœ… | âœ… | | | Grouping | âœ… | âœ… | âœ… | âœ… | âœ… | âœ… | | | Chart views | âœ… | âŒ | âŒ | âŒ | âŒ | âŒ | | | Table view | âœ… | âœ… | âœ… | âœ… | âœ… | âœ… | | | CSV data export | âœ… | âœ… | âœ… | âœ… | âœ… | âœ… | | | Proceeds | âœ… | âœ… | âŒ | âŒ | âŒ | âœ… | | | Taxes | âœ… | âœ… | âŒ | âŒ | âŒ | âœ… |

Time ranges

When using the Adapty calendar to set a time range for a chart, you have several convenient options to choose from. These options determine the date range displayed on the x-axis of the charts, allowing you to focus on specific periods of time. Here are the available quick options for time ranges:

- **Last 7 days:** Displays data for the most recent 7-day period.
- **Last month:** Shows data from the current date to the same day in the previous month.
- **Last 28 days:** Useful for tracking weekly subscription products, as it covers the last four weeks.
- **Last 3 months:** *Displays data from the current date to three months ago.
- **Last 6 months:** *Shows data from the current date to six months ago.
- **Last year:** Displays data from the current date to one year ago.
- **Previous month:** Covers the full calendar month before the current month.
- **This month:** *Shows data from the 1st day of the current month until today.
- **This quarter:** *Displays data from the 1st day of the current quarter until today.
- **This year:** Covers data from the 1st day of the current year until today.

In addition to these predefined options, you can also select the **Custom** option to set a specific time period of your choice. This flexibility allows you to analyze your data in more granular detail or focus on specific events or campaigns.

Apart from selecting specific time ranges, you can also adjust the time scale of charts. By choosing a day timescale, you can view the most detailed level of data, while opting for lower resolutions such as week, month, quarter, or year, allows you to identify longer-term trends.

For charts, the time scale determines the scale of the grid on the x-axis and the resolution at which the data is displayed. The available time scale options include day, week, month, quarter, or year. Choosing a specific time scale allows you to view the data at different resolutions, helping you identify shorter-term or longer-term trends.

When analyzing cohorts in Adapty, you choose the cohort length, which determines the size of each cohort and the grouping of users based on a specific time period. The time frame in cohorts analysis refers to the duration for which you want to analyze user behavior, and it helps define the boundaries of the cohorts.

In LTV (lifetime value) analysis, the cohort length is chosen in the grouping settings and determines the time period over which you want to calculate the lifetime value of users. Similar to cohorts analysis, the cohort length in LTV analysis is independent of the time frame displayed on the x-axis of the charts.

We have 2 formats of date and time - American and European. You can set one of them in your Adapty account as described here.

2023-07-10at_20.56.292x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Please also note that all charts in Adapy analytics are displayed in UTC time.

Data comparison

To analyze the dynamics of your app's metrics, you can utilize the comparison feature located next to the calendar. It offers a convenient way to compare your metrics with the previous period, although you also have the flexibility to customize the comparison range based on your specific requirements.

Here's how you can interpret the insights provided by the comparison feature:

- **Comparison Display:** After selecting the comparison period, you can toggle between displaying the comparison on the chart or as a numerical value only.
- **Difference Indicator:** The comparison shows the variance between your current result and the result from the previous period. Higher values are indicated in green, while lower values are indicated in red.
- **Chart Visualization:** If you have no grouping or only one grouping selected, the comparison will be displayed on the chart as well. You can choose from different chart types such as area, line, or column to visually highlight the differences.
- **Detailed Tooltip:** Hovering over the chart will reveal a tooltip with additional details, allowing you to examine the specifics of the comparison.
- **Multiple Grouping and Comparison:** If you have multiple grouping options enabled, you can view multiple comparisons simultaneously on a single chart. This feature is available specifically for column charts.

2023-07-10at_20.59.372x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Filtering and grouping

Filters play a crucial role in refining the data displayed in charts by including only the information that matches specific attributes. This feature becomes especially handy when you wish to examine the performance of a particular property, such as a specific country or product identifier.

By grouping chart data, you can analyze the individual components that make up the chart totals. This is particularly valuable when you want to evaluate the performance of particular properties.

It's important to note that certain charts may not support every type of filtering or grouping. To determine the compatibility of filters and grouping options with each chart, you can refer to the corresponding information provided on the respective chart description page.

In Advanced Analytics, you have access to the following filtering and grouping options, empowering you to refine your data analysis:

| | Filtering | Grouping | Description | | :---------- | :------- | :------ | :---------- | | Attribution | âœ… | âœ… | Filter or group metrics by Attribution fields like Status, Channel, Campaign, Adgroup, Adset, and Creative. | | Renewal Status | âŒ | âœ… | Group metrics by the product's renewal status, which indicates whether or not the subscription will be renewed in the next period. | | Period | âŒ | âœ… | Group metrics by the period. | | Country | âœ… | âœ… | Filter or group metrics by user's store country (if available, otherwise country is inferred using currency code or device's IP). | | Paywall (Deprecated) | âœ… | âœ… | Filter or group metrics by paywalls that are associated with purchases. (Deprecated legacy option) | | Paywall | âœ… | âœ… | Filter or group metrics by paywalls that are associated with purchases. | | A/B Test | âœ… | âŒ | Filter or group metrics by A/B teststhat are associated with purchases. | | Placement | âœ… | âœ… | Filter or group metrics by placementthat are associated with purchases. | | Store | âœ… | âœ… | Filter or group metrics by store (iOS/Android). | | Product | âœ… | âœ… | Filter or group metrics by product (both subscriptions and one-time purchases are available). | | Duration | âœ… | âœ… | Filter or group metrics by product's duration. |

2023-09-12at_15.15.152x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Chart views

The Analytics section provides you with the flexibility to view each chart in different visual representations, such as stacked column, stacked area, line, 100% stacked column, and 100% stacked area. By selecting the appropriate view, you can effectively communicate the information displayed on the chart and enhance your data analysis experience.

To change the view of a chart, simply locate and select the view dropdown menu, and then choose the desired representation option

2023-07-10at_20.53.002x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Table view

In addition to the chart view, Adapty also provides a table view for each chart. The table view presents the underlying data used to generate the chart in a tabular format, allowing users to view and analyze the data in a more granular way. The table view is a useful tool for users who prefer to work with data in a more structured format or need to export the data for further analysis outside of Adapty.

CSV data export

If you want to analyze the underlying data used to generate a chart, cohorts analysis, funnels, retentions, or conversion analytics, you can easily export it in CSV format by clicking the **Export** button. This feature allows you to access the raw data used to create the respective analysis. You can manipulate this data in spreadsheet applications or other tools to perform further analysis.

2023-07-10at_20.49.152x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Store commission and taxes

One crucial aspect of revenue calculation is the inclusion of taxes (which can vary based on the user's store account country) and store commission fees. Adapty currently supports commission and tax calculation for both App Store and Play Store.

In the charts tab of the Analytics section, Adapty introduces a dropdown field with three display options.

2023-07-10at_20.51.382x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The dropdown allows you to choose how the revenue is displayed in the chart. The available options are as follows:

Gross revenue

This option displays the total revenue, including taxes and commission fees from both App Store / Play Store. It represents the complete revenue generated by transactions before any deductions.

Proceeds after store commission

This option displays the revenue amount after deducting the store commission fee.
It represents the revenue that remains after the App Store / Play Store cuts its commission fees from the gross revenue. Taxes are not deducted in this display option.

Apple and Google take up to 30% of the price paid by the customers as a fee. For the apps included in Small Business Program (i.e. the app makes less than $1m per year), the fee is always 15%. The rest of the apps (>$1m per year) pay 30% by default and 15% for subscriptions that are consecutively renewed for at least a year. For detailed information on how Adapty calculates store commission fees, please refer to the corresponding documentation for App Store and Play Store.

Proceeds after store commission and taxes

This option displays the revenue amount after deducting both the store commission fee and taxes.
It represents the net revenue received by the app after accounting for both store's commission and applicable taxes. We consider the VAT rate of the user's store account country when calculating taxes. Please consider that Adapty follows the logic that for Apple taxes are applied to the post-commission revenue from a transaction, while Google applies taxes to the full amount (before store commissions are reduced from the revenue).

It's important to note that this revenue dropdown applies to various revenue-related charts, such as Revenue, MRR (Monthly recurring revenue), ARR (Annual recurring revenue), ARPU (Average revenue per user), and ARPPU (Average revenue per paying user). These charts provide valuable insights into revenue-related information for your app.

---

title: "Analytics charts" description: ""

metadataTitle: ""

Adapty's charts provide a comprehensive view of your app's performance metrics, enabling you to quickly and easily track key data points like user engagement, retention, and revenue.

To access any of Adapty's charts, navigate to the Charts section in the Analytics dashboard.

2023-07-10at_17.27.102x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty charts are built using real-time data, so you can always stay up-to-date with the latest metrics. We also offer advanced controls that allow you to filter and group data by a range of criteria, including country, paywall, and product, giving you even greater flexibility and insight.

Adapty offers a wide range of charts to help you analyze and visualize your app's performance data. Here are the charts currently available in Adapty:

Subscription and in-app charts

- Revenue: Track your app's total revenue, broken down by date or other criteria.
- MRR (Monthly recurring revenue): See how your app's recurring revenue is performing month-over-month.
- ARR (Annual recurring revenue): Track your app's recurring revenue on an annual basis.
- ARPU (Average revenue per user): Measure how much revenue your app is generating per user.
- ARPPU (Average revenue per paying user): Measure how much revenue your app is generating per paying user.
- Installs: Monitor the number of app installs over a specific period.
- Active subscriptions: Keep track of the number of active subscriptions in your app.
- New subscriptions: Analyze the number of new subscriptions generated in a specific period.
- Subscription renewal cancelled: Track the number of subscriptions that were canceled before renewal.
- Expired (churned) subscriptions: See the number of expired subscriptions or users who have churned.
- Non-subscriptions: Analyze the revenue generated by non-subscription in-app purchases.

Trials charts

- Active trials: Monitor the number of active free trials in your app.
- New trials: Analyze the number of new trials generated in a specific period.
- Trial renewal cancelled: Track the number of trials that were canceled before renewal.
- Expired (churned) trials: See the number of expired trials.

Issues charts

- Grace period: Analyze how long users are in the grace period after a failed payment.
- Billing issues: Track the number of failed payments, chargebacks, and other billing issues.
- Refund events: Monitor the number of refunds processed by your app.
- Refund money: Analyze the amount of revenue lost due to refunds.

Each of these charts provides a unique perspective on your app's performance, helping you identify trends, track progress, and make data-driven decisions.

Adapty's charts are designed to be highly customizable, giving you complete control over how you view and analyze your app's performance data. To get the most out of our charts, check our Analytics controls documentation, which provides a detailed overview of all the features and functionality available.

---

title: "Revenue" description: ""

metadataTitle: ""

The revenue chart displays the total revenue earned from both subscriptions and one-time purchases, minus the revenue that was refunded later. The Revenue chart is an essential tool to monitor the financial performance of the app.

2023-05-04at_16.53.552x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The Adapty revenue chart calculates the total revenue generated by the app minus the refunded revenue. This includes both new revenue, which comes from a customer's first paid transaction, such as new paid subscriptions, initial non-consumable and consumable purchases, and trial-to-paid conversions, as well as renewal revenue, which comes from subsequent paid transactions such as paid subscription renewals, promotional offers for existing subscribers, additional non-renewing subscription purchases, and more.

The calculation is done before deducting the store's fees and taxes. Revenue from each transaction is attributed to the period in which the transaction occurred, which can lead to period-over-period fluctuations based on the mix of subscription durations being started or renewed. It's important to note that refunds are treated as negative revenue and attributed to the day they occurred on (not to the day the subscription began).

Revenue = Total amount billed to customers - Revenue from refunded purchases/subscriptions.

For example, there were 5 monthly $10 subs, 1 yearly $100 sub, and 10 one-time $50 purchases today,

revenue = 5*$10 + 1*$100 + 10*$50 = $650

After calculating the total revenue earned from in-app purchases, Adapty's Revenue chart provides an estimate of the expected proceeds by deducting the store's commission fee and taxes. The commission fee is calculated based on the gross revenue, while taxes are deducted before the commission fee is applied. This ensures that the displayed proceeds reflect the net revenue received by the app after accounting for both the store's commission and applicable taxes. To learn more about how Adapty calculates commission fees, taxes, and estimates the expected proceeds, refer to the relevant documentation.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

Revenue chart usage

The Revenue chart is a valuable tool to track the financial performance of the app. By analyzing the chart's data, you can gain insight into the growth trajectory of your app over time. One useful approach is to switch to a monthly resolution and observe the last 12 months of revenue to assess the app's overall trend. It is also helpful to analyze the mix of new and renewal revenue to identify where growth is coming from. You can further segment the data by different dimensions, such as product or user type, to gain deeper insights into the app's financial performance. Overall, the Revenue chart is an essential metric for understanding the financial health of an app and optimizing its revenue strategy.

Similar metrics

In addition to Revenue, Adapty also provides metrics for other revenue-related events, such as MRR, ARR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- MRR
- ARR
- ARPU
- ARPPU

---

title: "Revenue" description: ""

metadataTitle: ""

The revenue chart displays the total revenue earned from both subscriptions and one-time purchases, minus the revenue that was refunded later. The Revenue chart is an essential tool to monitor the financial performance of the app.

2023-05-04at_16.53.552x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The Adapty revenue chart calculates the total revenue generated by the app minus the refunded revenue. This includes both new revenue, which comes from a customer's first paid transaction, such as new paid subscriptions, initial non-consumable and consumable purchases, and trial-to-paid conversions, as well as renewal revenue, which comes from subsequent paid transactions such as paid subscription renewals, promotional offers for existing subscribers, additional non-renewing subscription purchases, and more.

The calculation is done before deducting the store's fees and taxes. Revenue from each transaction is attributed to the period in which the transaction occurred, which can lead to period-over-period fluctuations based on the mix of subscription durations being started or renewed. It's important to note that refunds are treated as negative revenue and attributed to the day they occurred on (not to the day the subscription began).

Revenue = Total amount billed to customers - Revenue from refunded purchases/subscriptions.

For example, there were 5 monthly $10 subs, 1 yearly $100 sub, and 10 one-time $50 purchases today,
revenue = 5*$10 + 1*$100 + 10*$50 = $650

After calculating the total revenue earned from in-app purchases, Adapty's Revenue chart provides an estimate of the expected proceeds by deducting the store's commission fee and taxes. The commission fee is calculated based on the gross revenue, while taxes are deducted before the commission fee is applied. This ensures that the displayed proceeds reflect the net revenue received by the app after accounting for both the store's commission and applicable taxes. To learn more about how Adapty calculates commission fees, taxes, and estimates the expected proceeds, refer to the relevant documentation.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

Revenue chart usage

The Revenue chart is a valuable tool to track the financial performance of the app. By analyzing the chart's data, you can gain insight into the growth trajectory of your app over time. One useful approach is to switch to a monthly resolution and observe the last 12 months of revenue to assess the app's overall trend. It is also helpful to analyze the mix of new and renewal revenue to identify where growth is coming from. You can further segment the data by different dimensions, such as product or user type, to gain deeper insights into the app's financial performance. Overall, the Revenue chart is an essential metric for understanding the financial health of an app and optimizing its revenue strategy.

Similar metrics

In addition to Revenue, Adapty also provides metrics for other revenue-related events, such as MRR, ARR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- MRR
- ARR
- ARPU
- ARPPU

---

title: "MRR" description: ""

metadataTitle: ""

The Monthly recurring revenue (MRR) chart displays the normalized revenue generated by your active paid subscriptions on a monthly basis. This chart enables you to understand your business's velocity and size, regardless of the fluctuations that may arise from varying subscription durations.

2023-05-11at_17.32.242x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the predictable and recurring revenue components of your subscription business using the following formula:

$$MRR_i = \sum_{subscriptions} \frac{P_s * N_s}{D_{sm}}$$

Where:
Ps - subscription price
Ns - number of active paid subscriptions for this subscription. Adapty considers any paid subscription that has not yet expired as an active subscription.
Dsm - subscription duration in months (0.25 for weekly subscriptions)

It is important to note that Adapty does not include non-recurring subscriptions, consumable, or one-time purchases in the calculation of Monthly Recurring Revenue (MRR). This is because these types of purchases do not represent predictable and recurring revenue.

Basically, MRR shows revenue from all active subscriptions normalized to one month. For example, for a yearly subscription, instead of counting full revenue from the start, revenue is split into 12 equal parts which are evenly spread across 12 month period.

E.g. if there are 2 active yearly subscriptions with price $240 and 10 monthly subscriptions with a price $30,
MRR = (2 * $240 / 12) + (10 * $30 / 1) + (20 * $10 / 0.25) = $1140

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution Ad group, attribution Ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

MRR chart usage

MRR is a crucial metric for businesses that rely on recurring subscription revenue. It not only captures the size of your subscriber base but also standardizes different subscription durations to a common denominator (monthly recurring revenue). By doing so, MRR provides a real velocity metric for your business, making it easier to track your growth trajectory accurately.

To leverage MRR effectively, segment your subscriber cohorts by their first purchase month and change the resolution to monthly. By doing this, you can create a stacked area chart that reveals how monthly subscriber cohorts have translated over time. This approach enables you to identify trends and patterns in your subscriber base, making it easier to adjust your business strategy and optimize your products and marketing efforts accordingly.

Similar metrics

In addition to MRR, Adapty also provides metrics for other revenue-related events, such as Revenue, ARR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- ARR
- ARPU
- ARPPU

---

title: "MRR" description: ""

metadataTitle: ""

The Monthly recurring revenue (MRR) chart displays the normalized revenue generated by your active paid subscriptions on a monthly basis. This chart enables you to understand your business's velocity and size, regardless of the fluctuations that may arise from varying subscription durations.

2023-05-11at_17.32.242x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the predictable and recurring revenue components of your subscription business using the following formula:

$$MRR_i = \sum_{subscriptions} \frac{P_s * N_s}{D_{sm}}$$

Where:
Ps - subscription price
Ns - number of active paid subscriptions for this subscription. Adapty considers any paid subscription that has not yet expired as an active subscription.
Dsm - subscription duration in months (0.25 for weekly subscriptions)

It is important to note that Adapty does not include non-recurring subscriptions, consumable, or one-time purchases in the calculation of Monthly Recurring Revenue (MRR). This is because these types of purchases do not represent predictable and recurring revenue.

Basically, MRR shows revenue from all active subscriptions normalized to one month. For example, for a yearly subscription, instead of counting full revenue from the start, revenue is split into 12 equal parts which are evenly spread across 12 month period.

E.g. if there are 2 active yearly subscriptions with price $240 and 10 monthly subscriptions with a price $30,
MRR = (2 * $240 / 12) + (10 * $30 / 1) + (20 * $10 / 0.25) = $1140

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution Ad group, attribution Ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

MRR chart usage

MRR is a crucial metric for businesses that rely on recurring subscription revenue. It not only captures the size of your subscriber base but also standardizes different subscription durations to a common denominator (monthly recurring revenue). By doing so, MRR provides a real velocity metric for your business, making it easier to track your growth trajectory accurately.

To leverage MRR effectively, segment your subscriber cohorts by their first purchase month and change the resolution to monthly. By doing this, you can create a stacked area chart that reveals how monthly subscriber cohorts have translated over time. This approach enables you to identify trends and patterns in your subscriber base, making it easier to adjust your business strategy and optimize your products and marketing efforts accordingly.

Similar metrics

In addition to MRR, Adapty also provides metrics for other revenue-related events, such as Revenue, ARR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- ARR
- ARPU
- ARPPU

---

title: "ARR" description: ""

metadataTitle: ""

The Annual recurring revenue chart shows revenue from all active auto-renewable subscriptions normalized to one year. The chart considers any paid, unexpired subscription as active. ARR is a crucial metric for tracking your subscription business's growth and predicting future revenue.

2023-05-08at_17.57.252x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The Adapty ARR chart calculates the total revenue generated by the app. Any paid, unexpired subscription is considered active. ARR includes normalized annual revenue from all active paid subscriptions â€“ even if their auto-renew status is currently disabled. This also means non-recurring subscriptions, consumable, or one-time purchases included in ARR calculation. The metric is calculated before the store's fee.

ARR = sum of ((Ps * Ns / Dsy)

where Ps - subscription price,

Ns - number of active paid subscriptions for this subscription,

Dsy - subscription duration in years (1/12 for monthly and ~1/52 for weekly subscriptions).

This metric is only useful when annual subscriptions are the biggest chunk of your sales.
For example, there are 2 active annual subscriptions with a price of $240, and 10 monthly subscriptions with a price of $30, and 20 weekly subscriptions with a price of $10,
ARR = (2 _ $240 / 1) + (10 _ $30 / (1/12)) + (20*$10 / (1/52)) = $14480

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARR chart usage

The Annual Recurring Revenue (ARR) chart is a valuable tool for measuring the growth and scale of your subscription-based business. It's a widely-used metric that provides a normalized view of your recurring revenue over a 12-month period. To gain a better understanding of what's driving your ARR, you can segment your data by key subscriber segments such as Store or Product Duration. By doing so, you can identify which segments are driving the most revenue, and optimize your business strategy accordingly.

Similar metrics

In addition to the ARR chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPU
- ARPPU

---

title: "ARR" description: ""

metadataTitle: ""

The Annual recurring revenue chart shows revenue from all active auto-renewable subscriptions normalized to one year. The chart considers any paid, unexpired subscription as active. ARR is a crucial metric for tracking your subscription business's growth and predicting future revenue.

.2023-05-08at_17.57.252x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The Adapty ARR chart calculates the total revenue generated by the app. Any paid, unexpired subscription is considered active. ARR includes normalized annual revenue from all active paid subscriptions â€“ even if their auto-renew status is currently disabled. This also means non-recurring subscriptions, consumable, or one-time purchases included in ARR calculation. The metric is calculated before the store's fee.

ARR = sum of ((Ps * Ns / Dsy)

where Ps - subscription price,

Ns - number of active paid subscriptions for this subscription,

Dsy - subscription duration in years (1/12 for monthly and ~1/52 for weekly subscriptions).

This metric is only useful when annual subscriptions are the biggest chunk of your sales.
For example, there are 2 active annual subscriptions with a price of $240, and 10 monthly subscriptions with a price of $30, and 20 weekly subscriptions with a price of $10,
ARR = (2 _ $240 / 1) + (10 _ $30 / (1/12)) + (20*$10 / (1/52)) = $14480

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARR chart usage

The Annual Recurring Revenue (ARR) chart is a valuable tool for measuring the growth and scale of your subscription-based business. It's a widely-used metric that provides a normalized view of your recurring revenue over a 12-month period. To gain a better understanding of what's driving your ARR, you can segment your data by key subscriber segments such as Store or Product Duration. By doing so, you can identify which segments are driving the most revenue, and optimize your business strategy accordingly.

Similar metrics

In addition to the ARR chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARPU, and ARPPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPU
- ARPPU

---

title: "ARPU" description: ""

metadataTitle: ""

ARPU (average revenue per user) chart displays the average revenue generated per user for a given period. This metric is calculated by dividing the total revenue generated by a cohort of customers by the number of users in that cohort.

.2023-05-09at_14.58.472x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the ARPU chart by dividing the total revenue earned over a given period by the number of non-unique users (installs) during the same period.

ARPU = Revenue / Number of new users

For instance, if your app generated $10,000 in revenue over the course of a week, and had 2,000 non-unique users during that same period, the ARPU for that week would be $5 ($10,000/2,000).

The calculation is done before the store's fee and the refund amount is excluded from the revenue.

Available filters and grouping

- âœ… Filter by: Attribution, country, and store.
- âœ… Group by: Country, store, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARPU chart usage

ARPU chart usage is beneficial for businesses to track their overall revenue generation and understand how much revenue is being generated per user. By analyzing the ARPU chart, businesses can identify trends, patterns, and areas of improvement to optimize their revenue generation strategies. This can help businesses make data-driven decisions to increase their user engagement, target their marketing efforts, and improve their overall monetization strategy.

Similar metrics

In addition to the ARPU chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARR, and ARPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPPU
- ARR

---

title: "ARPU" description: ""

metadataTitle: ""

ARPU (average revenue per user) chart displays the average revenue generated per user for a given period. This metric is calculated by dividing the total revenue generated by a cohort of customers by the number of users in that cohort.

2023-05-09at_14.58.472x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the ARPU chart by dividing the total revenue earned over a given period by the number of non-unique users (installs) during the same period.

ARPU = Revenue / Number of new users

For instance, if your app generated $10,000 in revenue over the course of a week, and had 2,000 non-unique users during that same period, the ARPU for that week would be $5 ($10,000/2,000).

The calculation is done before the store's fee and the refund amount is excluded from the revenue.

Available filters and grouping

- âœ… Filter by: Attribution, country, and store.
- âœ… Group by: Country, store, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARPU chart usage

ARPU chart usage is beneficial for businesses to track their overall revenue generation and understand how much revenue is being generated per user. By analyzing the ARPU chart, businesses can identify trends, patterns, and areas of improvement to optimize their revenue generation strategies. This can help businesses make data-driven decisions to increase their user engagement, target their marketing efforts, and improve their overall monetization strategy.

Similar metrics

In addition to the ARPU chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARR, and ARPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPPU
- ARR

---

title: "ARPPU" description: ""

metadataTitle: ""

ARPPU

The Average revenue per paying user (ARPPU) chart displays the average revenue per paid user. It displays the actual revenue generated by paying customers, divided by the number of customers, minus refunds.

2023-05-09at_14.10.022x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The ARPPU chart is calculated as:

ARPPU = Revenue / Number of users who paid

For example, if your app generated $10,000 in revenue over a given period and had 500 paying customers during that time, the ARPPU would be calculated as $10,000 / 500 = $20. This means that on average, each paying customer generated $20 in revenue during the selected period.

It's important to note that the ARPPU value displayed represents the sum of total revenue divided by the total number of paying customers. The calculation is done before the store's fee and the refund amount is excluded from the revenue. As one user can pay more than one time during the whole period, the total ARPPU value may be higher than the daily value of ARPPU. This metric provides valuable insights into the revenue-generating capabilities of your app's paying user base and can help you optimize your pricing and subscription strategy to maximize revenue generation.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARPPU chart usage

The ARPPU chart in Adapty is a powerful tool that can help businesses understand the revenue generated by each paying user. By analyzing this metric over time, businesses can identify which channels, networks, and campaigns attract the most valuable customers. This information can be used to optimize marketing strategies and increase revenue by targeting high-value customers. With Adapty's ARPPU chart, businesses can make data-driven decisions that improve their bottom line and drive long-term success.

Similar metrics

In addition to the ARPPU chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARR, and ARPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPU
- ARR

---

title: "ARPPU" description: ""

metadataTitle: ""

ARPPU

The Average revenue per paying user (ARPPU) chart displays the average revenue per paid user. It displays the actual revenue generated by paying customers, divided by the number of customers, minus refunds.

Calculation

The ARPPU chart is calculated as:

ARPPU = Revenue / Number of users who paid

For example, if your app generated $10,000 in revenue over a given period and had 500 paying customers during that time, the ARPPU would be calculated as $10,000 / 500 = $20. This means that on average, each paying customer generated $20 in revenue during the selected period.

It's important to note that the ARPPU value displayed represents the sum of total revenue divided by the total number of paying customers. The calculation is done before the store's fee and the refund amount is excluded from the revenue. As one user can pay more than one time during the whole period, the total ARPPU value may be higher than the daily value of ARPPU. This metric provides valuable insights into the revenue-generating capabilities of your app's paying user base and can help you optimize your pricing and subscription strategy to maximize revenue generation.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

ARPPU chart usage

The ARPPU chart in Adapty is a powerful tool that can help businesses understand the revenue generated by each paying user. By analyzing this metric over time, businesses can identify which channels, networks, and campaigns attract the most valuable customers. This information can be used to optimize marketing strategies and increase revenue by targeting high-value customers. With Adapty's ARPPU chart, businesses can make data-driven decisions that improve their bottom line and drive long-term success.

Similar metrics

In addition to the ARPPU chart, Adapty also provides metrics for other revenue-related events, such as Revenue, MRR, ARR, and ARPU. To learn more about these revenue-related metrics, please refer to the following documentation guides:

- Revenue
- MRR
- ARPU
- ARR

---

title: "Installs" description: ""

metadataTitle: ""

The Installs chart shows the total number of users who have installed the app for the first time, as well as any reinstalls by existing users. This includes multiple installations by the same user on different devices. Please note that incomplete downloads or installations that are canceled before completion are not counted toward the install count.

Calculation

The calculation logic for the Installs chart provided by Adapty involves counting the total number of times the app has been installed by both new and existing users, including any reinstalls on different devices. However, incomplete installations or downloads that were canceled before completion are not included in the total count.

Please note that If your app has in-app user authentication, the Installs chart may also include the count of new logged-in users who have accessed your app multiple times. For more details please check our SDK documentation for identifying users

Available filters and grouping

- âœ… Filter by: Attribution, country, and store.
- âœ… Group by: Country, store, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Installs chart usage

The Installs chart provides a useful metric to track the overall growth of the user base. By analyzing the chart, you can gain insights into the number of new users who have installed their app for the first time, as well as any reinstalls by existing users. This information can help to identify trends and patterns in user acquisition over time, and make informed decisions about marketing and promotional activities.

---

title: "Installs" description: ""

metadataTitle: ""

The Installs chart shows the total number of users who have installed the app for the first time, as well as any reinstalls by existing users. This includes multiple installations by the same user on different devices. Please note that incomplete downloads or installations that are canceled before completion are not counted toward the install count.

Calculation

The calculation logic for the Installs chart provided by Adapty involves counting the total number of times the app has been installed by both new and existing users, including any reinstalls on different devices. However, incomplete installations or downloads that were canceled before completion are not included in the total count.

Please note that If your app has in-app user authentication, the Installs chart may also include the count of new logged-in users who have accessed your app multiple times. For more details please check our SDK documentation for identifying users

Available filters and grouping

- âœ… Filter by: Attribution, country, and store.
- âœ… Group by: Country, store, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Installs chart usage

The Installs chart provides a useful metric to track the overall growth of the user base. By analyzing the chart, you can gain insights into the number of new users who have installed their app for the first time, as well as any reinstalls by existing users. This information can help to identify trends and patterns in user acquisition over time, and make informed decisions about marketing and promotional activities.

---

title: "Active subscriptions" description: ""

metadataTitle: ""

The Active subscriptions chart displays the amount of unique paid subscriptions that have not yet expired at the end of each selected period. It includes regular (unexpired) in-app subscriptions that started and are currently active and excludes both free trials and subscriptions with canceled renewals.

Calculation

Adapty's Active Subscriptions calculation logic counts the number of paid, unexpired subscriptions at the end of a given period. At a daily resolution, the amount of Active subscriptions represents the number of unexpired

subscriptions at the end of that day. Therefore, the count of Active subscriptions for a given day indicates the number of unexpired subscriptions at the end of that day. At a monthly resolution, the count of Active Subscriptions represents the number of unexpired subscriptions at the end of that month. Note that, a paid subscription without a grace period is considered expired once its next renewal date has passed without a successful renewal.

For example, if there were 500 active subscriptions at the end of last month, 50 new subscriptions were started this month, and 25 subscriptions expired this month, then there are 525 active subscriptions at the end of this month.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Active subscriptions chart usage

The Active subscriptions chart is useful to get valuable insights into the number of recurring, individual paid users from your app. This metric serves as a proxy for the size and growth potential of a business. Combining Active Subscriptions with filters and grouping helps you to gain a deeper understanding of their paid subscriber base composition, making it a powerful tool for data analysis.

Similar metrics

In addition to Active subscriptions, Adapty also provides metrics for other subscription-related events, such as new subscriptions, subscriptions renewal canceled, expired subscriptions, and non-subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation:

- Churned (expired) subscriptions
- Cancelled subscriptions
- Non-subscriptions

---

title: "New subscriptions" description: ""

metadataTitle: ""

The New subscriptions chart displays the amount of new (first-time activated) subscriptions in your app. This metric shows the number of new subscriptions starting in a specific time period, including both subscriptions that start from scratch and free trials that convert into paid subscriptions. It does not include subscription renewals or subscriptions that have been restarted.

2023-05-05at_16.53.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty's calculation logic for the New subscriptions chart counts the number of new (first-time activated) subscriptions during a given period, including both subscriptions that start from scratch and free trials that convert into paid subscriptions. At a daily resolution, the count of new subscriptions represents the number of new subscriptions activated on that day. Therefore, the count of new subscriptions for a given day indicates the number of new subscriptions activated on that day. At a monthly resolution, the count of new subscriptions represents the number of new subscriptions activated during that month.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

New subscriptions chart usage

The new subscriptions chart provides valuable insights into the number of newly acquired, individual paid users for your app. This metric is crucial for assessing the growth potential of your business. By applying filters and grouping to the new subscription data, you can enhance your understanding of the composition of your new subscriber base. This chart serves as a powerful tool for in-depth data analysis in terms of new user acquisition.

Similar metrics

In addition to New subscriptions, Adapty also provides metrics for other subscription-related events, such as active subscriptions, subscriptions renewal canceled, expired subscriptions, and non-subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation guides:

- Active subscriptions
- Churned (expired) subscriptions
- Cancelled subscriptions
- Non-subscriptions

---

title: "Non-subscriptions" description: ""

metadataTitle: ""

The Non-subscriptions chart displays the number of in-app purchases such as consumables, non-consumables, and non-renewing subscriptions. The chart doesn't include renewable payments. The chart shows the total count of these types of in-app purchases and can help you track user behavior and engagement over time.

2023-05-12at_12.41.002x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty's calculation logic for the Non-subscriptions chart involves counting the number of in-app purchases made by users that are classified as consumables, non-consumables, and non-renewing subscriptions. This chart excludes renewable payments such as auto-renewing subscriptions.

- Consumables are items that users can purchase multiple times, such as fish food in a fishing app, or extra in-game currency.
- Non-consumables are items that users can purchase once and use forever, such as a race track in a game app, or ad-free versions of an app.
- Non-renewing subscriptions are subscriptions that expire after a set period of time and do not renew automatically, such as a one-year subscription to a catalog of archived articles. The content of this in-app purchase can be static, but the subscription will not renew automatically once it expires.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, and store.
- âœ… Group by: Product, country, store, paywall, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Non-subscriptions chart usage

The Non-subscriptions chart is an important tool for app developers to gain insights into the types of in-app purchases made by users, including consumables, non-consumables, and non-renewing subscriptions. By tracking this metric over time, you can better understand user behavior and engagement with their app. Using the Non-subscriptions chart with filters and grouping, you can dive deeper into their users' purchase patterns and preferences, helping them to optimize pricing strategies and improve overall user satisfaction. The Non-subscriptions chart is an essential tool for app to make data-driven decisions that ultimately lead to a better user experience and increased revenue.

Similar metrics

In addition to non-subscriptions, Adapty also provides metrics for other subscription-related events, such as active subscriptions, new subscriptions, subscriptions renewal canceled, and expired subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation guides:

- Active subscriptions
- New subscriptions
- Churned (expired) subscriptions
- Cancelled subscriptions

---

title: "Non-subscriptions" description: ""

metadataTitle: ""

The Non-subscriptions chart displays the number of in-app purchases such as consumables, non-consumables, and non-renewing subscriptions. The chart doesn't include renewable payments. The chart shows the total count of these types of in-app purchases and can help you track user behavior and engagement over time.

2023-05-12at_12.41.002x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty's calculation logic for the Non-subscriptions chart involves counting the number of in-app purchases made by users that are classified as consumables, non-consumables, and non-renewing subscriptions. This chart excludes renewable payments such as auto-renewing subscriptions.

- Consumables are items that users can purchase multiple times, such as fish food in a fishing app, or extra in-game currency.
- Non-consumables are items that users can purchase once and use forever, such as a race track in a game app, or ad-free versions of an app.
- Non-renewing subscriptions are subscriptions that expire after a set period of time and do not renew automatically, such as a one-year subscription to a catalog of archived articles. The content of this in-app purchase can be static, but the subscription will not renew automatically once it expires.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, and store.
- âœ… Group by: Product, country, store, paywall, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Non-subscriptions chart usage

The Non-subscriptions chart is an important tool for app developers to gain insights into the types of in-app purchases made by users, including consumables, non-consumables, and non-renewing subscriptions. By tracking this metric over time, you can better understand user behavior and engagement with their app. Using the Non-subscriptions chart with filters and grouping, you can dive deeper into their users' purchase patterns and preferences, helping them to optimize pricing strategies and improve overall user satisfaction. The Non-subscriptions chart is an essential tool for app to make data-driven decisions that ultimately lead to a better user experience and increased revenue.

Similar metrics

In addition to non-subscriptions, Adapty also provides metrics for other subscription-related events, such as active subscriptions, new subscriptions, subscriptions renewal canceled, and expired subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation guides:

- Active subscriptions
- New subscriptions
- Churned (expired) subscriptions
- Cancelled subscriptions

---

title: "Subscriptions renewal cancelled" description: ""

metadataTitle: ""

The Subscriptions renewal canceled chart displays the number of subscriptions that have had their auto-renew status switched off (canceled by the user). When a subscription's auto-renew status is turned off, it means that the subscription will not renew automatically for the next period. However, the user still retains access to the app's premium features until the end of the current period.

2023-05-08at_15.22.252x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty's calculation logic for the subscriptions renewal cancelled chart involves counting the number of subscriptions that have had their auto-renewal status switched off during a given period. This includes subscriptions that were canceled by the user and will not renew automatically for the next period.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Subscriptions renewal canceled chart usage

The Active subscriptions chart is useful to get valuable insights into the number of recurring, individual paid users from your app. This metric serves as a proxy for the size and growth potential of a business. Combining active subscriptions with filters and grouping helps you to gain a deeper understanding of their paid subscriber base composition, making it a powerful tool for data analysis.

Similar metrics

In addition to Subscription renewal canceled chart, Adapty also provides metrics for other subscription-related events, such as active subscriptions, new subscriptions, expired subscriptions, and non-subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation guides:

- Active subscriptions
- Churned (expired) subscriptions
- New subscriptions
- Non-subscriptions

---

title: "Churned (expired) subscriptions" description: ""

metadataTitle: ""

The churned (expired) subscriptions chart displays the number of subscriptions that have expired, meaning that the user no longer has access to the premium features of the app. Typically, this occurs when the user decides to stop paying at the end of the subscription period for the app or encounters a billing issue.

2023-05-08at_15.09.592x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

The churned (expired) subscriptions chart calculation logic for Adapty involves counting the number of subscriptions that have expired during a given period. This includes users who have decided to stop paying for the app or those who have experienced billing issues. To obtain this chart, the number of expired subscriptions should be counted daily or monthly. At a daily resolution, the count of expired subscriptions represents the number of subscriptions that expired on that day, while at a monthly resolution, it represents the number of expired subscriptions during that month.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Expiration reason, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

Churned subscriptions chart usage

The Churned (expired) subscriptions chart is a useful metric to gain insights into the number of users who have stopped paying for the app or have experienced billing issues during a specific period. This metric provides information on the number of users who have churned, which can be used to identify trends in user behavior and billing issues. By combining the Churned subscription chart with filters and grouping, app developers or business owners can gain a deeper understanding of their user base and analyze the reasons for churn.

Similar metrics

In addition to Churned subscriptions, Adapty also provides metrics for other subscription-related events, such as active subscriptions, new subscriptions, subscriptions renewal canceled, and non-subscriptions. To learn more about these subscriptions-related metrics, please refer to the following documentation guides:

- Active subscriptions

- New subscriptions
- Cancelled subscriptions
- Non-subscriptions

---

title: "Active trials" description: ""

metadataTitle: ""

The active trials chart in Adapty displays the number of unexpired free trials that are currently active at the end of a given period. Active means subscriptions that have not yet expired, and therefore, users still have access to the paid features of the app.

Calculation

Adapty calculates the number of active trials in a given period by referring to the count of unexpired free trials by the end of that period. This count remains the same until the trial expires, irrespective of its auto-renew status. At a daily resolution, the count of Active Trials represents the number of unexpired trials by the end of that day.

For example, if 100 trials were active yesterday, 10 new trials were activated today, and 5 trials have expired today, then there are 105 active trials today.

However, at a monthly resolution, the count of Active Trials represents the number of unexpired trials by the end of that month.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Active trials chart usage

This chart provides valuable insights into the effectiveness of your app's trial offers and allows you to monitor the number of users currently taking advantage of your free trial periods. By leveraging the insights provided by the Active Trials chart, you can optimize your app's free trial strategy and maximize user engagement and revenue generation. With Adapty's powerful analytics and monitoring tools, you'll have everything you need to make data-driven decisions that drive your app's success.

Similar metrics

In addition to Active Trials, Adapty also provides metrics for other trial-related events, such as New trials, Trial Renewal cancelled, and Expired trials. To learn more about these trial-related metrics, please refer to the following documentation:

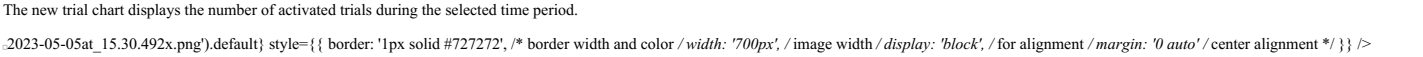
- New trials
- Trial renewal cancelled
- Expired trials

---

title: "New trials" description: ""

metadataTitle: ""

The new trial chart displays the number of activated trials during the selected time period.



Calculation

Adapty's calculation of the new trials refers to the number of trials initiated during a specific period. Adapty tracks the number of trials started within the selected period, regardless of their status (expired or active) at the end of the period.

For example, if you select a monthly period and 50 users start a trial during that month, then the number of new trials initiated during that period would be 50. Similarly, if you choose a daily resolution, Adapty tracks the number of trials started each day, regardless of their status at the end of the day.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

New trials chart usage

The New trials chart is a powerful tool for tracking the effectiveness of your app's promotional campaigns and user acquisition efforts. For instance, if you run a targeted ad campaign on social media or search engines, you can use the New trials chart to monitor the number of new trials initiated during the campaign period. By analyzing this data, you can determine the effectiveness of the campaign and make data-driven decisions to optimize your promotional strategies in the future.

Similar metrics

In addition to New Trials, Adapty also provides metrics for other trial-related events, such as Active trials, Trial renewal cancelled, and Expired trials. To learn more about these trial-related metrics, please refer to the following documentation:

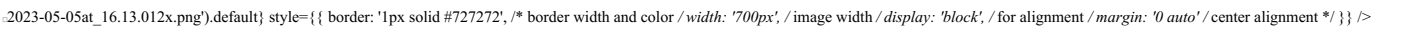
- Active trials
- Trial renewal cancelled
- Expired trials

---

title: "Trials renewal cancelled" description: ""

metadataTitle: ""

The Trials renewal cancelled chart displays the number of trials with canceled renewal (cancelled by user). When the renewal for the trial is disabled, this means that this trial won't be automatically converted to a paid subscription, yet the user still has premium features of the app until the end of the current period.



Calculation

Adapty calculates the number of Trials renewal cancelled during a specific period by counting the trials that users have canceled before the end of their trial period. This count remains unchanged regardless of the trial's auto-renewal status.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Renewal status, period, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Trials renewal canceled chart usage

The Trials renewal canceled chart is a useful tool that provides insights into the trial cancellation patterns of your app's users. By monitoring the number of users who have canceled their trial subscriptions during a specific period, you can optimize your app's trial offerings and enhance user satisfaction. By leveraging the insights provided by this chart, you can determine whether your trial strategy needs adjustment to encourage users to continue their subscriptions.

Similar metrics

In addition to the Trials renewal canceled chart, Adapty also provides metrics for other trial-related events, such as New trials, Active trials, and Expired trials. To learn more about these trial-related metrics, please refer to the following documentation:

- New trials
- Active trials
- Expired trials

---

title: "Expired (churned) trials" description: ""

metadataTitle: ""

The Expired (churned) trials chart displays the number of trials that have expired, leaving users without access to the app's premium features. In most cases, this occurs when users decide not to pay for the app or experience billing issues.

2023-05-05at_16.31.132x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the number of "Expired (Churned) Trials" during a specific period by counting the number of trials that have ended and users no longer have access to premium app features. This count remains unchanged regardless of the reason for the trial's expiration, such as a user's decision not to pay for the app or billing issues.

In addition, Adapty allows you to group the chart by expiration reason, such as user-initiated cancellations or billing issues. This grouping provides valuable insights into why users are churning and enables you to take proactive measures to prevent churn and optimize your app's success.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Expiration reason, product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Expired (churned) trials chart usage

The Expired (churned) trials chart is a valuable tool that provides insights into the number of trial periods that have ended, leaving users without access to premium app features. By monitoring the number of users who have churned during a specific period, you can identify patterns in user behavior and billing issues, and take proactive measures to reduce churn and improve user retention. By leveraging the insights provided by this chart, you can adjust your app's strategy and billing practices to encourage users to continue their subscriptions.

Similar metrics

In addition to the Trials renewal canceled chart, Adapty also provides metrics for other trial-related events, such as New Trials, Active Trials, and Expired Trials. To learn more about these trial-related metrics, please refer to the following documentation:

- New trials
- Active trials
- Trials renewal canceled

---

title: "Refund events" description: ""

metadataTitle: ""

The Refund events chart displays the number of refunded purchases and subscriptions. Adapty attributes refund events to the moment when the refund occured, meaning that any refunds related to a subscription will be counted on an actual date of the refund and not on the start date of the subscription.

2023-05-11at_18.36.262x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates refund events by tracking the number of refunded purchases and subscriptions over a selected period of time. Refunds are attributed to the moment when they occured (not to when the subscription started), and any refunds that occur during the selected time period will be included in the refund events count. It's important to note that refunds related to trials are not included in this count, as trials have no revenue associated with them and cannot be refunded.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Refund events chart usage

The Refund Events chart in Adapty provides businesses with valuable insights into the number of refunded purchases and subscriptions over a selected period of time. By monitoring the number of refunds, businesses can identify any trends or issues related to refunds and make data-driven decisions to optimize their revenue streams. This chart is particularly useful for identifying any recurring refund issues and taking steps to address them.

Similar metrics

In addition to the Billing Issue chart, Adapty also provides metrics for other issues-related events, such as Refund money, Billing issue, and Grace period. To learn more about these issue-related metrics, please refer to the following documentation:

- Refund money
- Billing issue
- Grace period

---

title: "Refund money" description: ""

metadataTitle: ""

The refund money chart displays the amount of money that was refunded for the selected period of time. Adapty attributes refund money to the moment when the refund was issued and the revenue decreases in the period when the refund occured.

2023-05-09at_11.54.462x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Calculation

Adapty calculates the refund money chart considering only transactions that generate revenue, such as new paid subscriptions, renewals, and one-time purchases. Free trials, which do not generate revenue and cannot be refunded, are excluded from the transaction count. The refund amount is attributed to the date when the refund was made (not to when the subscription started), and the revenue decrease caused by the refunded subscription is reflected in the period in which the refund was processed.

It's important to note that Adapty calculates the refund amount before the store's fee is deducted.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Refund Money chart usage

The Refund money chart in Adapty helps businesses track the financial impact of refunds on their revenue. By monitoring refund amounts over time, you can identify patterns in customer behavior and make informed decisions to improve your revenue and reduce refund requests. With the refund money chart, you have a powerful tool to make data-driven decisions that optimize your business's bottom line.

Similar metrics

In addition to the Refund money chart, Adapty also provides metrics for other issues-related events, such as Billing events, Billing issue, and Grace period. To learn more about these issue-related metrics, please refer to the following documentation:

- Refund events
- Billing issue
- Grace period

---

title: "Grace period" description: ""

metadataTitle: ""

The Grace period chart displays the number of subscriptions that have entered the grace period state due to a billing issue. During this period, the subscription remains active while the store tries to receive payment from the subscriber. If payment is not successfully received before the grace period ends, the subscription enters the billing issue state.

2023-07-11at_15.20.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Calculation

Adapty calculates the Grace period chart by tracking the number of subscriptions that have entered the Grace period state within a given time period. The Grace period begins when the subscription enters the billing issue state due to a payment failure and ends after a specified amount of time (6 days for weekly subscriptions, 16 days for all other subscriptions) or when payment is successfully received. The chart provides insights into the effectiveness of the grace period feature and can help identify potential issues with payment processing or subscription management.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Similar metrics

In addition to the Grace period chart, Adapty also provides metrics for other issues-related events, such as Refund events, Refund money, and Billing issue. To learn more about these issue-related metrics, please refer to the following documentation:

- Refund money
- Refund events
- Billing issue

---

title: "Billing issue" description: ""

metadataTitle: ""

The Billing issue chart displays the number of subscriptions that have entered the Billing Issue state. This state is typically triggered when the store, such as Apple or Google, is unable to receive payment from the subscriber for some reason. This could happen due to reasons such as an expired credit card or insufficient funds.

2023-07-11at_15.21.262x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Calculation

Adapty calculates the billing issue chart by tracking the number of subscriptions that have entered the billing issue state during a given period. A subscription enters the Billing Issue state when the store (e.g. Apple, Google) is unable to process payment from the subscriber for some reason, such as an expired credit card or insufficient funds. During the Billing Issue state, the subscription is not considered active, and if the Grace Period feature is enabled in the store settings, the subscription will only move to the Billing Issue state after the Grace Period has expired.

Available filters and grouping

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, paywall, duration, attribution status, attribution channel, attribution campaign, attribution ad group, attribution ad set, and attribution creative.

You can find more information about the available controls, filters, grouping options, and how to use them in the this documentation.

Billing issue chart usage

The Billing Issue chart can provide valuable insights into your app's subscription performance and revenue. By tracking the number of subscriptions in the Billing Issue state over time, you can identify patterns and potential issues related to payment processing and user payment information. This information can be used to optimize your app's payment process and reduce the likelihood of subscription cancellations due to payment issues. You can also use the chart to track the impact of changes to payment processing and billing information update flows.

Similar metrics

In addition to the Billing Issue chart, Adapty also provides metrics for other issues-related events, such as Refund events, Refund money, and Grace period. To learn more about these issue-related metrics, please refer to the following documentation:

- Refund money
- Refund events
- Grace period

---

title: "Lifetime Value (LTV)" description: ""

metadataTitle: ""

The Realized LTV (Lifetime value) per paying customer displays the revenue that a paying customer cohort actually generated after refunds have been deducted, divided by the number of paying customers in that cohort. Therefore, this chart tells you how much revenue you generate on average from each paying customer.

Adapty designs the LTV chart to answer several important questions about your app's revenue and customer behavior such as:

1. How much money does each cohort bring in over their lifetime with your app?
2. At what point in time does a cohort pay off?
3. How can you optimize your app's marketing and acquisition spend to attract valuable, high LTV customers?
4. How long does it take to recoup your investment in acquiring new customers?

The LTV chart works with the app data we gather through our SDK and in-app events.

With this information, you will be able to gain insights into how your subscriptions are performing and how much revenue is generated from your subscribers during a given period of time. You can use this information to make informed decisions about your subscription offerings, ad spending, and customer acquisition strategies. Additionally, the filters will allow you to segment the data by country, attribution, and other variables, giving you a more granular understanding of your customer base.

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

LTV by renewals

The *LTV by renewals *view presents data pertaining to the subscription period (P), specifically capturing the first instance when a customer makes a payment. In the case of a weekly subscription, this corresponds to the subsequent weekly subscription period.

LTV by days

The *LTV by days *view organizes and filters data based on daily, weekly, or monthly intervals. It provides insights into the total revenue generated by all users who installed the app on a specific day, week, or month, divided by the number of paying users during that same period. This view offers valuable insights into revenue tracking and enables a comprehensive understanding of user behavior over time.

Calculation

Realized LTV is calculated using the total revenue generated from each customer cohort, minus refunds.

LTV for the day/week/month = Revenue gained from all the paying users who installed the app on this day/week/month / the number of paying users who installed the app on this day/week/month

The LTV calculation includes upgrades, downgrades, and reactivations, such as when a user changes their subscription plan or pricing. It takes into account the revenue generated from the initial subscription and subsequent renewals based on the updated plan.

LTV chart usage

The LTV chart is a valuable tool in Adapty that provides insights into the long-term value of your customers. By analyzing customer behavior and purchase patterns over time, the LTV chart helps you understand the revenue generated by different customer segments or cohorts.

The LTV chart is particularly useful for identifying high-value customer segments, tracking the effectiveness of marketing campaigns, and evaluating the overall financial performance of your business. By understanding the lifetime value of your customers, you can make informed decisions about resource allocation, customer acquisition strategies, and customer retention initiatives.

In addition, the LTV chart can be used to compare different customer segments, assess the impact of product changes or pricing adjustments, and identify opportunities for upselling or cross-selling.

Available grouping and filtering

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Both filters and groupings can be applied to both the renewals and days views of the LTV chart, allowing you to drill down into specific cohorts and understand their behavior over time

- âœ… Filter by: Attribution, country, paywall, store, product, and duration.
- âœ… Group by: Product, country, store, duration, and by cohort day, week, month, and year.

You can find more information about the available controls, filters, grouping options, tax and commission controls, and how to use them in this documentation.

The Realized LTV chart in Adapty helps to gain valuable insights into customer behavior, optimize marketing strategies, track revenue performance, and make data-driven decisions to maximize the long-term value of their customers.

---

title: "Cohort analysis" description: ""

metadataTitle: ""

Adapty cohorts are designed to answer several important questions:

1. On what day does a cohort pay off?
2. How much money does the app earn for a specific cohort?
3. How much money can I spend to attract a paying customer?
4. How long does it take to recoup the ad spend?

Cohorts work with the app data we gather through SDK and store notifications and don't require any additional configuration from your side.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Cohorts by renewals or by days

You can analyze cohorts by renewals or by days. The control changes the headings of the columns. Consequently, the approach to analysis also changes.

Tracking **by days** provides valuable insights for budgeting and understanding payment timelines. This is which is particularly useful for tracking non-subscription products, such as consumables or one-time purchases. In this mode, the blue color in the table cells tends to be concentrated in the middle of the lines due to two key factors. Firstly, viewing cohorts by days allows for early visibility of payments associated with short duration products, while in the renewals view, they are grouped with monthly and yearly renewals. Secondly, delayed payments contribute to the distribution pattern, as some users pay later than expected.

Whereas tracking **by renewals** shows the retention and churn of the cohorts from one payment to another without consideration of the date. So late users who paid with any delay (it can be months) are added to the number of their subscription period. This approach doesn't reflect the situation of calendar earnings but is definitely more convenient to analyze the retention and churn of the cohorts and get insights from their behavior.

Choose your convenient mode or use them both for more conclusions and ideas.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

How Adapty builds cohorts

Let's see in the example of cohorts by renewals how the table is formed. To build cohorts, we use two measures: app installations and transactions (purchases). Every row of a cohort represents a specific time interval: from a day to a year. Each row starts with the number of users who installed the app during this interval and activated a subscription or made a lifetime/non-subscription product purchase.

Every next column in the row shows the number of users who renewed a subscription to this period. M3 stands for month 3 and means that subscribers had 3 consecutive renewals to this point, W7 stands for week 7, and Y2 stands for year 2. Sometimes you can see P2 in cohorts. P stands for Period of subscription. Adapty displays instead of W/M/Y when there are multiple products with different renewal periods present in the same cohort.

We use gradient colors to highlight differences in cohort values. The biggest numbers have more saturated colors.

In the image below you can see a typical cohort.

1. This cohort displays the data only for weekly products (mark #1).
2. It doesn't exclude proceeds and shows the revenue as absolute values (mark #2).
3. The time period we're working with is the last 6 months, and every cohort segment is 1 month long (mark #3).
4. The **Total** row (mark #4) displays the cumulative value for each period. $395K in the first cell of the **Total** row accumulates the first period (subscription activation) revenue from all months (Nov, Dec, and so on) until the end of the timeframe. The Total cell shows the number of customers who installed the app during the whole period.
5. The first column of the Nov 2023 row (mark #5) shows the first period (subscription activation) revenue of $37.7K from the customers who installed the app in Nov 2023. The number of customers who installed the app in Nov 2023 which is 95,129, is shown in the header column.
   The second column of the Nov 2023 row shows week 2 (subscriptions renewed to the 2nd week) revenue of $8,77K who installed the app in Nov 2023.
6. On the table, you can see the Total revenue, ARPU, ARPPU, and ARPAS (mark #6). You can read more about them a little further in this article.
7. You can configure the columns in the right part of the table using the **Columns** dropdown field (mark #7).
8. Above the table on the right (mark #8), there is also a dropdown field to calculate stores' commission fees and tax calculations for the specific cohort analyses. You can learn about how Adapty calculates store commission fees and taxes further in this article. After choosing the corresponding option from the dropdown the revenue data will be recalculated based on it.
9. On the right side of the table, you can see predicted revenue (Predicted Revenue) and predicted lifetime value (Predicted LTV) (mark #9). The **Predicted Revenue** field estimates the total revenue generated by a subscriber cohort within a specific timeframe, while the **Predicted LTV** field represents the anticipated value of each user in the cohort.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can hover on any cell in the cohort to view detailed metrics for this period.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The cells with oblique lines in the background are the periods that are not finished yet, so the values in them might be increased.

Filters, metrics, cohort segments, and export in CSV

Adapty offers a wide range of controls to help you gain valuable insights when looking into your cohorts' analyses. By default, Adapty builds cohorts based on the data from all purchases. It might be useful to filter all the products of the same duration or specific products. You can also use country, store, paywall, segment, and attribution data as a filter. You can find more information about the available controls, filters, grouping options, and how to use them in this documentation.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

On the right of the control panel, there's a button to export cohort data to CSV. You can then open it in Excel, or Google Sheets, or import it into your own analytical system.

There are 4 metrics that can be shown in cohorts: Subscriptions, Payers, Revenue, ARPU, ARPPU, and ARPAS. You can either display them as absolute values or as a relative change from the start of the cohort.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can set the date range for cohorts and choose the segment. The segment determines a timespan for each row of the cohort.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Subscriptions, payers, total revenue, ARPU, ARPPU and ARPAS

Subscriptions are the total count of active subscriptions, lifetime purchases, and non-subscription purchases made by a cohort within a selected timeframe. Monitoring this metric helps you understand customer behavior and the effectiveness of your offerings. This insight allows you to refine your product strategy, tailor marketing efforts, and optimize revenue streams.

Payers are the total number of users who made a purchase within a cohort. It helps you understand how many unique users contribute to your revenue. For apps with a significant amount of non-subscription purchases, this metric can highlight the true reach of your product offerings, showing whether a broad user base is making purchases or if revenue is driven by a smaller group of repeat buyers. Understanding the number of payers helps in assessing customer engagement, planning targeted marketing, and optimizing revenue strategies.

Total revenue is accumulated for a cohort within a selected timeframe (Nov 25, 2022 â€" May 24, 2023). It helps you to understand how much money you collected from users from a specific cohort and calculate ROAS. For example, if the ad spend for September 2022 was $10000, and the total proceeds for September 2022 cohort are $30000, ROAS=3:1.

ARPU is the average revenue per user. Itâ€™s calculated as total revenue / number of unique users. $60000 revenue / 5000 users = $12 ARPU. Itâ€™s helpful to compare this value to the cost of install (CPI) to understand the effectiveness of your marketing campaigns.

ARPPU is the average revenue per paying user. Itâ€™s calculated as total revenue / number of unique paying users. $60000 revenue / 1000 paying users = $60 ARPPU. It helps you to understand how much money brings you a paying customer on average.

ARPAS is the average revenue per active subscriber. Itâ€™s calculated as total revenue / number of active subscribers. By subscribers, we mean those who activated a trial period or subscription. $60000 revenue / 1500 subscribers = $40 ARPAS.

Commission fees and taxes

One important aspect of revenue calculation in cohorts is the inclusion of store commission fees and taxes (which can vary based on the user's store account country) and store commission fees. Adapty currently supports commission fee and taxes calculation for both App Store and Play Store in cohort analytics.
For more details on how Adapty calculates taxes and commissions in its analytics, please refer to our documentation.

Revenue vs Proceeds

Both Revenue and Proceeds are money metrics. You can think of Revenue as gross revenue and Proceeds as net revenue. Revenue doesn't account for App Store / Play Store fees, while Proceeds do. Therefore Proceeds are always less than Revenue (15%-30% less to be exact).

Apple and Google take up to 30% of the price paid by the customers as a fee. For the apps included in Small Business Program (i.e. the app makes less than $1m per year), the fee is always 15%. The rest of the apps (>$1m per year) pay 30% by default and 15% for subscriptions that are consecutively renewed for at least a year. This means 53+ renewals for weekly subscriptions, 13+ renewals for monthly subscriptions, and 2+ renewals for annual subscriptions.

Adapty automatically determines the fee for every transaction your customers make and calculates Proceeds based on it.

Prediction: Revenue and LTV

Predicted revenue is an estimated total revenue a cohort of paying subscribers is expected to generate within the selected period after cohort creation. It is calculated by multiplying the predicted LTV of the cohort by the predicted number of paying users within the cohort. For example, if the predicted LTV is $50 and there are 100 paying users in a cohort, the Predicted Revenue would be $5,000.

Predicted LTV is the estimated lifetime value per paying subscriber, representing the average revenue each paying subscriber is expected to generate within the selected period after cohort creation.

These predictions are done using machine learning (ML) models, which analyze historical customer data to identify patterns and make predictions about future revenue. For detailed documentation on Adapty's prediction models, please refer to our Prediction documentation.

Adapty cohorts provide detailed insights into user behavior and financial performance within your app. By analyzing cohorts based on renewals or days, you can determine when cohorts become profitable, track revenue, calculate average revenue per user, and understand the time it takes to recoup advertising spend. With customizable filters, metrics, and export options, Adapty empowers you to make data-driven decisions and optimize user acquisition and monetization strategies for maximum app success.

---

title: "Funnel analysis" description: ""

metadataTitle: ""

Adapty funnels are designed to assist you with such kinds of questions:

1. What percentage of installs is converted to paying clients?
2. What part of those who tried the product became loyal?
3. Which steps show high drop-off and need more attention?
4. Why do clients stop to pay?

With a funnel chart, you may also find more insights about user behavior setting filters and groups.

Funnels work with the data that we gather through SDK and store notifications and don't require any additional configuration from your side.

2022-06-24at_10.08.53.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Funnel chart step by step

Let's go through the elements of a funnel to understand how to read the user journey on the chart.

2022-06-23at_09.36.49.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The 1st column (1) is the number of installs. It is shown as an absolute value (2) of total installations (not unique users) and also as 100% - the largest input number for further conversions relative calculation. If a user deletes an app and then installs it again two separate installs will be counted.
A grey area nearby stands for transition parameters between steps. A conversion percent to the next step (Displayed paywall) is shown on a flag (3). Drop off percent and an absolute value of churn are shown below (4).

2022-06-23at_14.02.06.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The 2nd column (5) shows the number of users of the app who saw a paywall at least one time (6). They are taken only from those installs that happened in a selected period. If a user sees a paywall in the selected period but his install date is out of range his view is not counted.
There is also a percentage of such views taken from the 1st step (7). You may notice that this percent is equal to the grey flag (3) of the 1st step. This equality takes place only for these first steps.

We collect data for this step from all your paywalls that use the `logShowPaywall()` method. So please be sure to send every paywall view to Adapty using this method as described in the docs.

A grey area next to the 2nd column stands for transition. A conversion percent to the next step (Trial) is shown on a flag (8). Drop-off percent and the absolute value of churned customers after the paywall are shown below (9).

2022-06-23at_15.54.32.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The 3rd column (10) shows the number of trials activated on the paywalls by customers who installed the app within a selected period (11). If a filter is set to non-trial product(s) this value becomes zero and the column is empty.

See also a percent of trials taken from the 1st step, showing the conversion from installs to trials (12).
You may notice that this percent is not equal now to the grey flag (8) of the previous step conversion. This is because we compare the current value with the 1st step at the top of the chart and with the previous step on grey flags.
So a grey area next to the 3rd column shows a conversion percent to the next step (Paid) which is displayed on a flag (13). Drop-off percent and absolute value of churned customers during a trial period are shown below (14).

2022-06-23at15.54.32-_2.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Starting from the trial you can hover on the step to see churn reasons.

2022-06-23at_17.36.08.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The 4th column shows the number of activated subscriptions (15). For products without trials, this number includes direct subscriptions from a paywall. For products with trials, it contains the number of trials converted into paid subscriptions. If you have both types of products, with trial and without, it will be a sum of both.

The percent at the top shows the conversion from installs (16).
The percent on a grey flag shows conversion to the next step (renewal to the 2nd period) (17).
Drop off before the renewal to the 2nd period percent and absolute value are shown below the conversion (18).

2022-06-23at_15.54.32-3.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

This step starts a sequence of steps with a similar structure. After the 2nd renewal comes the 3rd, then the 4th, etc. If there is enough data in your app history you may see dozens of periods using the horizontal scroll. The logic for these steps remains the same:

- percent from installs at the top,
- percent from the previous step at the bottom,
- the absolute amount of renewal at the top,
- the absolute amount of churn at the bottom,
- a hover for churn reasons pop-up.

Table view, filters and CSV export

A funnel chart is enriched with data in a table to provide handy material for your work with numbers.

This table repeats the approach of the funnel with some amendments.
There are columns that show data on all steps except for the step of the 1st paid subscription.
Instead of this one, there are two separate: Install -> Paid and Trial -> Paid. They display a core point of conversion when a free user becomes paying.

It may seem that there is a product type division: Install -> Paid column shows only products without trials while the column Trial -> Paid contains only products with trials. But that's not exactly the way it works. Because we also consider those users whose trial has expired and they purchase a product with a trial like it doesn't have it at all.

Diving deeper into numbers you will find filtering powerful tools for new hypotheses.
Feel free to set conditions in different dimensions. Collect true insights based on data.
Variate:

1. Product type - economy, length, etc
2. Time range.
3. Country segmentation.
4. Traffic attribution.
5. Store.

Select Absolute #, Relative %, or both to view only necessary data.

Finally, on the right of the control panel, there's a button to export funnel data to CSV. You can then open it in Excel, or Google Sheets, or import it into your own analytical system.

:::warning Be sure to indicate that your app is included in Small Business Program in Adapty General Settings. :::

---

title: "Retention analysis" description: ""

metadataTitle: ""

Retention charts can help with the following questions:

1. How does your app retain clients from period to period?
2. What products are more attractive and hold better?
3. What groups of users are more loyal?
4. Which level of retention can be used as a benchmark for growth?
5. And of course, how can you save money investing in the attracted audience instead of capturing new.

You'll find valuable insights about user behavior setting filters and groups.

Retention is performed with the data that we gather through SDK and store notifications and don't require any additional configuration from your side.

How do we calculate retention?

Observing the retention chart, you see how the number of users depends on the step they take: trial (if the checkbox "show trials" is checked), the 1st payment, the 2nd payment, etc. Let's specify what users are counted when you choose a date range for the retention chart.
For example, you've selected the last 3 months in the calendar, and the checkbox "show trials" is unchecked. This means we count only those who have had their 1st subscription during the last 3 months. If the checkbox "show trials" is checked and the 3 last months are selected in the calendar we count all those who have had their trials during the last 3 months. For these subscribers, we show the absolute retention for the Nth step as the number of those who have had the Nth payment. And we calculate a relative value of retention for the Nth step as a ratio of the absolute amount of the Nth payment to the total amount of subscriptions (or trials) during the selected time range.

:::info Retention changes retrospectively

Regardless of when you check the chart, the baseline number (100%) remains the same for the selected period of time. Meanwhile, the retention to the next period may grow over time. For example, for a Monthly subscription, if there are 20 first purchases made between Dec 1 and Dec 31, it is expected that retention to the second period will grow throughout January (and possibly even after) while users will be entering the next subscription period in time or later for some reasons (e.g. grace period). :::

Retention opportunities

Let's see how to get more from the Adapty retention feature.
Having not only a pure passion for numbers but more willingly seeing real business value after implementing analytical results, we may think about the purposes first. With a deep dive into chart features, it would be nice to clear up the impact this data can have.
So let's keep in a glance together WHY and HOW.

1 - work with the audience.
First of all, retention is about the target audience, its preferences, and whether your product meets their expectations or not during the consuming lifetime. If you have wondered how to measure the core relationship of your business that generates money - retention is at your service.
Such a measurement benefits because it's usually cheaper to sell to your customer than to a stranger. And this cost is low for two reasons: less effort to sell and higher average check. So it might be a good idea to invest in your subscribers' loyalty when retention goes down.

2 - work with the product.
The second reason WHY is that retention charts show the actual consuming lifetime of your product and let you forecast in long term. And if you want to improve, correct the job that delivers the product to change its lifetime, and then forecast again to become closer to your business targets. Such updates may be a part of a strategic vision working together with a forecasting routine. And yes, this process never ends because we all run fast to be at the same place in a constantly changing environment.

3 - work with the market.
Moving faster than the main competitors is good but sometimes jumping out of the ordinary race may bring more benefits. When you analyze the behavior of users in different countries and stores, some local peculiarities can open outstanding insights and new opportunities for the business. Cultural and market context can be analyzed from the perspective of retention to be later used for segmentation and further development. For example, you may find blue water in some regions and grow there faster.

The usage of retention data is of course, not limited to this basic interpretation but it may be a good start if you want to get real value fast.

Curves, table view, filters and CSV export

Now when we are on the same page with retention purposes and basic ways of interpretation let's go through the tools that make it all handy.
The core of the retention feature in Adapty is the chart. It shows how retention level depends on the steps of a customer's lifetime.
The steps are shown on the horizontal axis: Trial, Paid (the 1st subscription), P2 (the 2nd subscription), P3, P4, etc
Please mind that the axis starts with the Trial step only when the checkbox "Show trials" is selected.

For data calculation, this checkbox works as follows. When "Show trials" is selected and the axis starts with the Trial step, you see only scenarios that contain trials, no transactions directly from installs are shown and the step Paid contains only transactions that come from trials. When "Show trials" is not selected, and the axis starts with a Paid step, this first step contains all first transactions including both from trials and directly from installs.

When you hover over the chart, a pop-up with a data summary is displayed. And if you hover over a column in the table below, you also see a summary pop-up with relevant data on the chart.
The table contains the same grouping and filters chosen for the chart.

Feel free to combine filters and grouping for advanced analysis. Collect true insights based on data.
Variate:

1. Product type.
2. Duration.
3. Time range.
4. Country.
5. Traffic attribution.
6. Store.

Use #Absolute and %Relative control to view the necessary data.

Finally, on the right of the control panel, there's a button to export funnel data to CSV. You can then open it in Excel, or Google Sheets, or import it into your own analytical system to continue analysis and forecasting in your preferred environment.

:::warning Be sure to indicate that your app is included in Small Business Program in Adapty General Settings. :::

---

title: "Conversion analysis" description: ""

metadataTitle: ""

While funnels show the overall picture and retention is focused on working with users' loyalty, the conversion feature is a handy tool for tracking the effectiveness at every point over time.

Conversions assist with the following questions:

1. How do your app conversions change over time? Are there any seasonal trends?
2. How conversions are changed in the moment of marketing activities or some other new circumstances?
3. How do different regions react to your app updates?
4. What types of products convert better over time?

You could find valuable insights about user behavior setting filters and groups.

Conversion is performed with the data we gather through SDK and store notifications and doesn't require any additional configuration from your side.

Main controls and charts

It's a common practice to measure success in money, e.g. tracking revenue. But sometimes it's important to see how your business works as a system at different financial scales over time. And here conversions come on stage.
There is usually a number of marketing activities, technical updates, and external events that may impact the changes in user behavior. To validate this inspect how conversions change over days, months and years.

The main steps control is to the left of the chart. It contains a list of conversions to track.

In general the logic of a conversion value X->Y for a particular day is that we use for a ratio the number of those who started their X state on the selected day and the number of those who then converted later (no matter when exactly) to the Y state, so conversions for the particular day are all associated with the date when a user activated an opportunity to convert to Y, which is the moment when he activated X. Using these we calculate a conversion = (Y / X) * 100%.

Please see below each conversion explanation with an example for your reference.

1. Install -> Paid
   If Đ¥ - the number of installs (the same for all products because there is no product chosen at the stage of installation) for the selected date, and Y - the amount of those among Đ¥ who paid (any possible time) for the 1st subscription (directly, without trial), then Conversion = (Y / X) _ 100%. For example, we had 100 installs on the 1st of January and among them, 20 subscribed the same week. On the 8th of January we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more installs of the 1st on January bought the 1st subscription without trial by the end of January. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had installed the app on the 1st of January converted to the 1st subscription without trial by the current moment.

2. Install -> Trial
   If Đ¥ - the number of installs (the same for all products because there is no product chosen at the stage of installation) for the selected date, and Y - the amount of those among Đ¥ who activated a trial (any possible time), then Conversion = (Y / X) _ 100%. For example, we had 100 installs on the 1st of January and among them 20 activated a trial the same week. On the 8th of January, we open the chart and see the conversion of the 1st of January (20 / 100) _ 100% = 20%. Then 30 more installs on the 1st of January activated a trial by the end of January. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had installed the app on the 1st of January converted to the trial by the current moment.

3. Trial -> Paid
   If Đ¥ - the number of trials taken during the selected date, and Y - the number of subscriptions after these trials taken any time later, then Conversion = (Y / X) _ 100%. For example, we had 100 trials taken on the 1st of January and among them, 20 bought a subscription the same week. On the 8th of January, we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more trials of the 1st of January paid by the end of January. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had activated the trial on the 1st of January converted to the 1st subscription by the current moment.

4. Paid -> 2nd Period
   If Đ¥ - the number of the 1st subscriptions taken during the selected date, and Y - the amount of the 2nd subscriptions taken after them any time later (normally the product duration time though grace period cases are also counted), then Conversion = (Y / X) _ 100%. For example, we had 100 1st subscriptions of various products on the 1st of January and among them 20 renewed in a week. On the 8th of January, we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of February. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 1st payment on the 1st of January converted to the 2nd subscription by the current moment.

5. 2nd Period -> 3rd Period
   If Đ¥ - the number of the 2nd subscriptions taken during the selected date, and Y - the amount of the 3rd subscriptions taken after them any time later (normally the product duration time though grace period cases are also counted), than Conversion = (Y / X) _ 100%. For example, we had 100 2nd subscriptions of various products on the 1st of January and among them 20 renewed in a week. On the 8th of January we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st on January renewed by the start of February. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 2nd payment on the 1st of January converted to the 3rd subscription by the current moment.

6. 3rd Period -> 4th Period
   If Đ¥ - the number of the 3rd subscriptions taken during the selected date, and Y - the amount of the 4th subscriptions taken after them any time later (normally the product duration time though grace period cases are also counted), then Conversion = (Y / X) _ 100%. For example, we had 100 3rd subscriptions of various products on the 1st of January and among them 20 renewed in a week. On the 8th of January, we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of February. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 3rd payment on the 1st of January converted to the 4th subscription by the current moment.

7. 4th Period -> 5 the Period
   If Đ¥ - the number of the 4th subscriptions taken during the selected date, and Y - the amount of the 5th subscriptions taken after them any time later (normally the product duration time though grace period cases are also counted), then Conversion = (Y / X) _ 100%. For example, we had 100 4th subscriptions of various products on the 1st of January and among them 20 renewed in a week. On the 8th of January, we open the chart and see the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of February. We open the chart on the 1st of February and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 4th payment on the 1st of January converted to the 5th subscription by the current moment.

8. 6 Months +
   If Đ¥ - the number of the 1st payments taken during the selected date, and Y - the amount of the renewals that happened after the 6 months since the selected date from those 1st payments, then Conversion = (Y / X) _ 100%. For example, we had 100 1st subscriptions of various products on the 1st of January and among them 20 renewed on the 1st week of July (the 25th payment). On the 8th of July, we open the chart and see

the conversion of the 1st of January = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of August (the 8th payment). We open the chart on the 1st of August and see that the conversion of the 1st of January = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 1st payment on the 1st of January converted to the period > 6 months with any number of payments by the current moment.

9. 1 Year +
   If Đ¥ - the number of the 1st payments taken during the selected date, and Y - the amount of the renewals that happened after the 12 months since the selected date from those 1st payments, then Conversion = (Y / X) _ 100%. For example, we had 100 1st subscriptions of various products on the 1st of January 2021 and among them 20 renewed on the 1st week of January 2022. On the 8th of January 2022, we open the chart and see the conversion of the 1st of January 2021 = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of February 2022. We open the chart on the 1st of February 2022 and see that the conversion of the 1st of January 2021 = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 1st payment on the 1st of January 2021 converted to the period > 12 months with any number of payments by the current moment.

10. 2 Years +
    If Đ¥ - the number of the 1st payments taken during the selected date, and Y - the amount of the renewals that happened after the 24 months since the selected date from those 1st payments, then Conversion = (Y / X) _ 100%. For example, we had 100 1st subscriptions of various products on the 1st of January 2020 and among them 20 renewed on the 1st week of January 2022. On the 8th of January 2022, we open the chart and see the conversion of the 1st of January 2021 = (20 / 100) _ 100% = 20%. Then 30 more subscribers of the 1st of January renewed by the start of February 2022. We open the chart on the 1st of February 2022 and see that the conversion of the 1st of January 2020 = ((20+30) / 100) * 100% = 50%. This number shows which part of those who had their 1st payment on the 1st of January 2020 converted to the period > 24 months with any number of payments by the current moment.

2022-08-02at_07.46.15.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The object for the analysis when the conversion is chosen is the chart. It performs how the conversion percentage changes over time. Using the date picker please select the quick options for the time period.
The chart usually contains several curves. Up to five of them are selected by default in the list of grouping and you may change the selection by choosing the checkboxes in the area to the right of the chart.
When you open the page for the first time the product duration is selected as a default grouping. Then your settings are saved in the cache and the next time you see the group you've recently selected.
The following groupings are available:

- Product
- Country
- Store
- Paywall
- Duration
- Marketing attribution

2022-08-01at_15.06.53.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

If a chosen date range is not enough to show any results, you may see a notification that offers a relevant date and an option to adjust the date range automatically so you may do it with one click.

2022-08-02at_08.05.02.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Table view, filters and CSV export

A comparison of the curves gives a bright picture, and to get more use the table view below the chart. The table is synchronized with the chart so hovering over a column you see the associated pop-up over the curves.

2022-08-02at_08.11.45.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The grouping that was mentioned above changes both the charts and the table. Set quick filter by product or use other advanced ones, including Product, Country, Store, Duration, Attribution.

2022-07-25at_22.24.38.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We know that it's important to have an option to work with numbers the way you like. So on the right of the control panel, there's a button to export funnel data to CSV. You can then open it in Excel, or Google Sheets, or import it into your own analytical system to continue analysis and forecasting in your preferred environment.

2022-07-25at_22.39.30.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::warning Be sure to indicate that your app is included in Small Business Program in Adapty General Settings. :::

---

title: "Reports" description: ""

metadataTitle: ""

Receive timely and relevant information straight to your inbox, including revenue, churn rate, active subscribers, active trials, and more â€“ the same metrics available in Charts. These reports can arrive on a daily, weekly or monthly basis and show dynamics comparing the most recent period to the one that came before it.

The data we send in the reports is based on what you have configured your **Overview** page meaning metrics, their order, reporting timezone and revenue type.

You have the flexibility to choose the level of detail you prefer for your reports: summary or per-app. A summary report is a single email containing aggregated information on all of your apps (or the subset of them that you have selected). A per-app report, in contrast, will only have the data for a single chosen app. We recommend enabling summary reports for all apps and per-app reports for recently released or high-priority apps, as well as those you are personally responsible for.

Regardless of the level of detail chosen, email reports are delivered to your inbox at 9 AM in your local time zone: daily reports arrive each day, weekly reports arrive on Mondays, and monthly reports arrive on the first day of the month. Each report includes current data along with comparisons to the previous period (e.g., for today's daily report, it compares data from yesterday and the day before; for today's weekly report, it compares data from last week and the week before, etc.).

Rest assured, whichever reports you select, you'll receive the most up-to-date and accurate information directly in your inbox.

Enable reports

1. Open the **Account** section in the Adapty top menu.
2. Under the **Email reports** section, choose the types of reports you wish to receive â€“ daily, weekly, and/or monthly.

/ width: '700px', / image width / display: 'block', / for alignment / margin: '0 auto' / center alignment */ }} />

2. Customize each report type by selecting the relevant apps. For this, click the **Edit** button.

reportsapps.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the report window, choose the apps you want to include.
4. Finally, click the **Save changes** button to apply your selections.

Set your time zone

1. Open the **Overview** section in the Adapty main menu.
2. Click the **Edit** button and choose your time zone.

/ width: '700px', / image width / display: 'block', / for alignment / margin: '0 auto' / center alignment */ }} />

3. Click the **Done** button to save.

---

title: "Reports" description: ""

metadataTitle: ""

Receive timely and relevant information straight to your inbox, including revenue, churn rate, active subscribers, active trials, and more â€“ the same metrics available in Charts. These reports can arrive on a daily, weekly or monthly basis and show dynamics comparing the most recent period to the one that came before it.

The data we send in the reports is based on what you have configured your **Overview** page meaning metrics, their order, reporting timezone and revenue type.

You have the flexibility to choose the level of detail you prefer for your reports: summary or per-app. A summary report is a single email containing aggregated information on all of your apps (or the subset of them that you have selected). A per-app report, in contrast, will only have the data for a single chosen app. We recommend enabling summary reports for all apps and per-app reports for recently released or high-priority apps, as well as those you are personally responsible for.

Regardless of the level of detail chosen, email reports are delivered to your inbox at 9 AM in your local time zone: daily reports arrive each day, weekly reports arrive on Mondays, and monthly reports arrive on the first day of the month. Each report includes current data along with comparisons to the previous period (e.g., for today's daily report, it compares data from yesterday and the day before; for today's weekly report, it compares data from last week and the week before, etc.).

Rest assured, whichever reports you select, you'll receive the most up-to-date and accurate information directly in your inbox.

Enable reports

1. Open the **Account** section in the Adapty top menu.
2. Under the **Email reports** section, choose the types of reports you wish to receive â€“ daily, weekly, and/or monthly.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

2. Customize each report type by selecting the relevant apps. For this, click the **Edit** button.

reportsapps.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

3. In the report window, choose the apps you want to include.
4. Finally, click the **Save changes** button to apply your selections.

Set your time zone

1. Open the **Overview** section in the Adapty main menu.
2. Click the **Edit** button and choose your time zone.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

3. Click the **Done** button to save.

---

title: "A/B test" description: ""

metadataTitle: ""

Are you looking to boost your in-app purchases and subscription revenue? One effective way to optimize your offerings is through A/B testing. With Adapty you can easily create and manage A/B tests for different pricing strategies, subscription lengths, trial durations, and more. In this guide, we'll walk you through the steps of creating A/B tests in the Adapty Dashboard and analyzing the results to make data-driven decisions about your in-app purchases and subscriptions. Whether you're new to A/B testing or looking to improve your existing strategies, this guide will provide you with the tools and insights you need to maximize your app's revenue potential.

Important

:::warning Be sure you send paywall views to Adapty using the `.logShowPaywall()`. Without this method, Adapty wouldn't be able to calculate views for the paywalls within the test, which will result in the irrelevant conversion stats. :::

How to create an A/B test

When creating a new A/B test, you must have at least two paywalls in it. Each paywall in the test will have a weight, which represents the relative amount of users it will receive during the test.

For example, if the first paywall has a weight of 70% and the second paywall weighs 30%, the first paywall will receive approximately 700 users out of 1000, and the second paywall will receive approximately 300 users. The sum of all weights must be 100%.

By configuring your A/B test options and assigning weights to each paywall, you can more effectively test different paywalls, enabling data-driven decisions for your business.

When it comes to creating A/B tests in Adapty, you have several options to choose from. Depending on your goals and needs, you can create a new A/B test directly from the A/B test section, from a specific placement page, or from a paywall page. Here's a step-by-step guide on how to create an A/B test using each of these options.

Creating A/B test from A/B test section

This way of creating A/B tests would require some adjustments on the SDK side. Before stepping to the further steps, check out our doc on displaying products.

To create a new A/B test from the **A/B tests** section:

1. Open the **A/B tests** item from the Adapty main menu.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

2. In the **A/B tests** window, click the **Create A/B test** button located at the top right side of the page.

ABtest.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

3. In the **Create the A/B test** window, enter the **A/B test name**. This is a mandatory field and should be something that helps you easily identify the test in the future. Choose a name that is descriptive and meaningful, so you can quickly recognize the test when reviewing its results later on.
4. Enter the **A/B test goal** for your further reference. Clearly defining the goal will help you stay focused on what you're trying to achieve with the test. The goal could be related to increasing subscription sign-ups, improving user engagement, or reducing bounce rates, among other objectives. By setting a specific goal, you can align your efforts and measure the success of the A/B test accurately.
5. Click the **Add paywall** button.

ABtest*with*data.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

3. Choose the paywalls you want to test and they will show in the **Create the A/B test** window.

testwith_Paywalls.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

4. You have two ways to save the new A/B test:
    1. You can choose to save the newly created paywall **as a draft**, which means the test won't be launched at once. You can run the test later from the placement or A/B test list. This option is suitable if you're not yet ready to make the test live and want to review and modify it further before implementation. If you choose this option, click the **Save as draft** button.
    2. Alternatively, you can choose to run the A/B test immediately by clicking the **Run A/B test** button. This will prompt you to select the placement and audience for which the A/B test will get live. Once you click the "**Run A/B test**" button, the A/B test will become active and live.
    To learn more about running A/B tests and the process involved, you can refer to the documentationon running A/B tests. You'll be able to monitor and track the test's performance using various metrics. For more information on these metrics, please refer to the metrics documentation.

Creating A/B test from the placement

Another way to create an A/B test is to do it from the Placement detail page. To get started, navigate to the detail page of the corresponding placement. Here, for each selected audience you have two options: you can choose to show any paywall to the users in that audience by selecting a paywall, or you can choose to run an A/B test by selecting the corresponding option.

If you're not familiar with Placements, you can learn more about them in our documentation.

For creating a new A/B test you should click on the **Create A/B test** button for the corresponding audience. The rest of the process is similar to creating an A/B test from the A/B test section.

You'll be prompted to choose whether you want to select an existing paywall group or create a new A/B test from scratch. If you choose to create a new A/B test, you can add your desired paywalls in the creation modal by clicking the "Add Paywall" button.

Once you've added your paywalls, you can assign weights to each option to control how often they appear during the test. You can also set a test goal to track your progress and determine which paywall performs best with your audience.

2023-04-25at_14.53.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' */ center alignment */ }} />

Once you've created your A/B test and saved the placement, it will be available in the corresponding placement of your app and displayed for the selected audience.

This means that users in the selected audience will see the paywalls included in the A/B test as they interact with your app. The weights assigned to each paywall option will determine how often each paywall appears during the test, allowing you to collect data on which paywalls perform best with your audience.

You can monitor the progress of your A/B test in the dashboard and make adjustments as needed to improve your results. One way to do this is by checking the metrics of the placement. You can learn more about placement metrics in this doc [insert link].

By setting up an A/B test from the Placement detail page, you'll be able to more easily test different paywalls with specific audiences and get valuable insights into what works best for your business.

---

title: "Run and stop A/B test" description: ""

metadataTitle: ""

Running A/B test in Adapty means adding it to a placement. In Adapty, you have two options to run an A/B test: from the **Placement** page or the **A/B test** page.

How to run the A/B test

In Adapty, you have two ways to run an A/B test: from the **Placement** page or the **A/B test** page.

To run your A/B test for the chosen audience and placement:

1. Open the **A/B tests** section from the Adapty main menu.

2. Click on the **Run** button next to the A/B test you want to begin. The **Run** button only shows for A/B tests that are neither **Live** nor **Completed**.
3. In the opened **Running A/B test** window, select a **Placement** from the drop-down list. This indicates where in your app the A/B test will display for the chosen audience. This list includes all placements you have in Adapty for your app.
4. From the **Audience** drop-down list, choose user segment for your A/B test. This list includes all segments you have in Adapty for your app.
   If you choose an audience that is a part of some other placement, it will automatically become a part of the chosen placement as well after you run the A/B test.
   Please note that every audience has a numerical **Priority** (starting from 1). If you add an audience with a priority lower than #1 to an A/B test, potential users for this A/B test may be directed to an audience with a higher priority instead, bypassing their participation in this A/B test. You can adjust audience priority in the placement itself.
5. After you've chosen the audience and placement for the A/B test, click the **Run** button to launch it.

After launching, you can then track its progress and view metrics on the A/B test metrics page. This will help you identify the better-performing variation and make informed decisions to enhance your app's performance. For more details on Adapty A/B test metrics, refer to Maths behind the A/B tests section.

How to stop the A/B test

When you choose to stop an A/B test, it means you have finished observing and analyzing the data. This step is essential for evaluating the test accurately and making informed decisions for future strategies. Stopping an A/B test is a crucial part of the testing process to optimize your outcomes effectively.

There are two options available to stop an A/B test: you can do so either from the **A/B tests** list page or the placement detail page. Regardless of whether you are on the A/B test list page or the placement detail page, both paths lead to the same flow.

1. Open the **A/B tests** section from the Adapty main menu and locate the A/B test you want to stop. Only tests that are currently running can be stopped.
2. Click on the **Stop** button next to the A/B test.
3. In the opened **Stop the A/B test** window, choose how to finish the A/B test. You have two options:

- Select a winner paywall from the list of tested paywalls.
  Based on the included paywall metrics such as revenue, probability to be best (abbreviated as **P2BB**), and revenue per 1K users, you can choose the winning paywall from the list of tested paywalls. By selecting this option, after stopping the A/B test for the selected placement and audience, the winning paywall will be displayed in the app. This allows you to optimize your app's performance by showcasing the most effective paywall to your users.
- Choose to stop the A/B test without selecting a winner paywall.
  For this, select the **Don't show a specific paywall for the audience** radio-button. In this case, for the selected placement and audience, no paywalls from that A/B test will be displayed in the app. This option is useful if you want to pause the display of any paywalls for that specific combination of placement and audience.

4. Click the **Stop and complete this A/B test** button.

Once the A/B test is stopped, it will no longer be active, and the paywalls will no longer be displayed to users. However, you can still access the A/B test results and metrics on the A/B test metrics page to analyze the data collected during the test.

:::note Stopping an A/B test is irreversible, and the test cannot be restarted once it has been stopped. Ensure that you have gathered sufficient data and insights before making the decision to stop an A/B test. :::

---

title: "A/B test results and metrics" description: ""

metadataTitle: ""

Discover important data and insights from our A/B tests, comparing different paywalls to see how they affect user behavior, engagement, and conversion rates. By looking at the metrics and results here, you can make smart choices and improve your app's performance. Dive into the data to find actionable insights and enhance your app's success.

A/B test results

Here are three metrics that Adapty provides for A/B test results:

Revenue: The revenue metric represents the total amount of money generated in USD from purchases and renewals. It includes both the initial purchase and subsequent subscription renewals. This metric helps you evaluate the financial performance of each A/B test variant and determine which option generates the highest revenue.

Probability to be best: Adapty utilizes a robust mathematical analysis framework to analyze A/B test results and provides a metric called Probability to be best. This metric assesses the likelihood that a particular variant is the best-performing option (in terms of its long-term revenue) among all the variants tested. The metric is expressed as a percentage value ranging from 1% to 100%. For detailed information on how Adapty calculates this metric, please refer to the documentation.The best performing option, determined by Revenue per 1K user, is highlighted in green and automatically selected as the default choice.

Revenue per 1K users: The revenue per 1K users metric calculates the average revenue generated per 1,000 users for each A/B test variant. This metric helps you understand the revenue efficiency of your variants, regardless of the total number of users. It allows you to compare the performance of different variants on a standardized scale and make informed decisions based on revenue generation efficiency.

**Prediction intervals for revenue 1K users:* *The revenue per 1K users metric also includes prediction intervals. These prediction intervals represent the range within which the true revenue per 1,000 users for a given variant is predicted to fall based on the available data and statistical analysis.

In the context of A/B testing, when analyzing the revenue generated by different variants, we calculate the average revenue per 1,000 users for each variant. Since revenue can vary among users, the prediction intervals provide a clear indication of the plausible values for the revenue per 1,000 users, taking into account the variability and uncertainty associated with the prediction process.

By incorporating prediction intervals into the revenue per 1K users metric, Adapty enables you to assess the revenue efficiency of your A/B test variants while considering the range of potential revenue outcomes. This information helps you make data-driven decisions and optimize your subscription strategy effectively, by taking into account the uncertainty in the prediction process and the plausible values for revenue per 1,000 users.

By analyzing these metrics provided by Adapty, you can gain insights into the financial performance, statistical significance, and revenue efficiency of your A/B test variants, enabling you to make data-driven decisions and optimize your subscription strategy effectively.

A/B test metrics

Adapty provides a comprehensive set of metrics to help you effectively measure the performance of your A/B test conducted on your paywall variations. These metrics are continuously updated in real-time, except for views, which are updated periodically. Understanding these metrics will help you assess the effectiveness of different variations and make data-driven decisions to optimize your paywall strategy.

A/B test metrics are available on the A/B test list, where you can gain an overview of the performance of all your A/B tests. This comprehensive view offers aggregated metrics for each test variation, enabling you to compare their performance and identify significant differences. For a more detailed analysis of each A/B test, you can access the A/B Test detail metrics. This section provides in-depth metrics specific to the selected A/B test, allowing you to delve into the performance of individual variations.

All metrics, except for views, are attributed to the product within the paywall.

Metrics controls

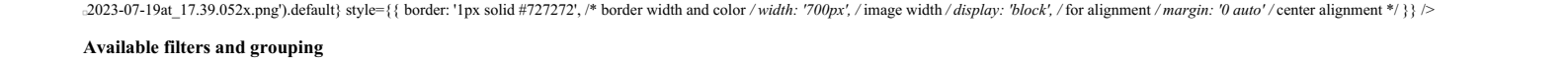
The system displays the metrics based on the selected time period and organizes them according to the left-side column parameter with three levels of indentation.

Profile install date filtration

The Filter metrics by install date checkbox enables the filtering of metrics based on the profile install date, instead of the default filters that use trial/purchase date for transactions or view date for paywall views. By selecting this checkbox, you can focus on measuring user acquisition performance for a specific period by aligning metrics with the profile install date. This option is useful for customizing the metrics analysis according to your specific needs.

Time ranges

You can choose from a range of time periods to analyze metrics data, allowing you to focus on specific durations such as days, weeks, months, or custom date ranges.



Available filters and grouping

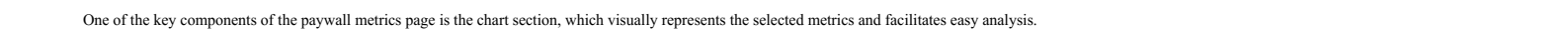
Adapty offers powerful tools for filtering and customizing metrics analysis to suit your needs. With Adapty's metrics page, you have access to various time ranges, grouping options, and filtering possibilities.

- âœ… Filter by: Audience, country, paywall, paywall state, paywall group, placement, country, store, product, and product store.
- âœ… Group by: Product and store.

You can find more information about the available controls, filters, grouping options, and how to use them for paywall analytics in this documentation.

Single metrics chart

One of the key components of the paywall metrics page is the chart section, which visually represents the selected metrics and facilitates easy analysis.



The chart section on the A/B test metrics page includes a horizontal bar chart that visually represents the chosen metric values. Each bar in the chart corresponds to a metric value and is proportional in size, making it easy to understand the data at a glance. The horizontal line indicates the timeframe being analyzed, and the vertical column displays the numeric values of the metrics. The total value of all the metric values is displayed next to the chart.

Additionally, clicking on the arrow icon in the top right corner of the chart section expands the view, displaying the selected metrics on the full line of the chart.

A/B test summary

Next to the single metrics chart, the A/B test details summary section is displayed, which includes information about the state, duration, placements, and other related details about the A/B test.

Metrics definitions

Here are the key metrics that are available for the A/B tests:

Revenue

Revenue represents the total amount of money generated in USD from purchases and renewals resulting from the A/B test. It includes the initial purchase and subsequent subscription renewals. The revenue metric is calculated before deducting the App Store or Play Store commission.

CR to purchases

The conversion rate to purchases measures the effectiveness of your A/B test in converting views into actual purchases. It is calculated by dividing the number of purchases by the number of views. For example, if you had 10 purchases and 100 views, the conversion rate to purchases would be 10%.

CR trials

The conversion rate (CR) to trials is the number of trials started from A/B test divided by the number of views. Conversion rate to trials measures the effectiveness of your A/B test in converting views into trial activations. It is calculated by dividing the number of trials started by the number of views.

Purchases

The purchases metric represents the total number of transactions made within the paywall resulting from the A/B test. It includes the following types of purchases:

- New purchases made on the paywall.
- Trial conversions of trials that were activated on the paywall.
  Downgrades, upgrades, and cross-grades of subscriptions on the paywall.
- Subscription restores on the paywall (e.g. when a subscription is expired without auto-renewal and is subsequently restored).

Please note that renewals are not included in the purchases metric.

Trials

The trials metric indicates the total number of activated trials resulting from the A/B test.

Trials cancelled

The trials canceled metric represents the number of trials in which auto-renewal has been switched off. This occurs when users manually unsubscribe from the trial.

Refunds

Refunds for the A/B test represent the number of refunded purchases and subscriptions specifically related to the tested paywall variations.

Views

Views are the number of views of the paywalls that the A/B test consits of. If the user visits the paywalls two times, this will be counted as two visits.

Unique views

Unique views are the number of unique views of the paywall. If the user visits the paywall two times, this will be counted as one unique view.

Probability to be the best

The Probability to be the best metric quantifies the likelihood that a specific paywall variant within an A/B test is the top-performing option among all the tested paywalls. It provides a numerical probability indicating the relative performance of each paywall. The metric is expressed as a percentage value ranging from 1% to 100%.

ARPPU (Average revenue per paying user)

ARPPU stands for Average Revenue Per Paying User resulting from the A/B test. It is calculated as the total revenue divided by the number of unique paying users. For example, if you have generated $15,000 in revenue from 1,000 paying users, the ARPPU would be $15.

ARPAS (Average revenue per active subscriber)

ARPAS is a metric that allows you to measure the average revenue generated per active subscriber from running the A/B test. It is calculated by dividing the total revenue by the number of subscribers who have activated a trial or subscription. For example, if the total revenue is $5,000 and you have 1,000 subscribers, the ARPAS would be $5. This metric helps assess the average monetization potential per subscriber.

Proceeds

The proceeds metric for the A/B test represents the actual amount of money received by the app owner in USD from purchases and renewals after deducting the applicable App Store / Play Store commission. It reflects the net revenue specifically associated with the paywall variations tested in the A/B test, contributing directly to the app's earnings. For more information on how proceeds are calculated, you can refer to the Adapty documentation.

Unique subscribers

The unique subscribers metric represents the count of distinct individuals who has subscribed or activated a trial through the paywall variations in the A/B test. It considers each subscriber only once, irrespective of the number of subscriptions or trials they initiate.

Unique paid subscribers

The unique paid subscribers metric represents the number of unique individuals who have successfully completed a purchase and become paying subscribers through the paywall variations in the A/B test.

Refund rate

The refund rate for the A/B test is calculated by dividing the number of refunds specifically associated with the paywall variations in the test by the number of first-time purchases (renewals are excluded). For instance, if there are 5 refunds and 1000 first-time purchases, the refund rate would be 0.5%.

Unique CR purchases

The unique conversion rate to purchases for the A/B test is calculated by dividing the number of purchases specifically associated with the paywall variations in the test by the number of unique views. For example, if there are 10 purchases and 100 unique views, the unique conversion rate to purchases would be 10%.

Unique CR trials

The unique conversion rate to trials for the A/B test is calculated by dividing the number of trials started specifically associated with the paywall variations in the test by the number of unique views. For example, if there are 30 trials started and 100 unique views, the unique conversion rate to trials would be 30%.

---

title: "Maths behind the A/B tests" description: ""

metadataTitle: ""

A/B testing is a powerful technique used to compare the performance of two different versions of a paywall. The ultimate goal is to determine which version is more effective based on the average revenue per user over a 12-month period. However, waiting for a full year to collect data and make decisions is impractical. Therefore, a 2-week revenue per user is used as a proxy metric, chosen based on historical data analysis to approximate the target metric. To achieve accurate and reliable results, it is crucial to employ a robust statistical method capable of handling diverse data types. Bayesian statistics, a popular approach in modern data analysis, provides a flexible and intuitive framework for A/B testing. By incorporating prior knowledge and updating it with new data, Bayesian methods allow for better decision-making under uncertainty. This document provides a comprehensive guide to the mathematical analysis employed by Adapty in evaluating A/B test results and providing valuable insights for data-driven decision-making.

Adapty's approach to statistical analysis

Adapty employs a comprehensive approach to statistical analysis in order to evaluate the performance of A/B tests and provide accurate and reliable insights. Our methodology consists of the following key steps:

1. **Metric definition:** To conduct an AB test successfully, you need to identify and define the key metric that aligns with the specific goals and objectives of the analysis. Adapty leveraged a huge amount of historical subscription app data to determine which fits the role of a proxy metric for the long-term goal of average revenue after 1 year - and it is ARPU after 14 days.
2. **Hypothesis formulation:** We create two hypotheses for the A/B test. The null hypothesis (H0) assumes that there is no significant difference between the control group (A) and the test group (B). The alternative hypothesis (H1) suggests that there is a significant difference between the two or more groups.
3. * *Distribution selection:* * We choose the best distribution family based on the data characteristics and the metric we observe. The most frequent choice here is log-normal distribution (taking into account zero values).
4. **Probability-to-be-best calculation:** Utilising the Bayesian approach to A/B testing, we calculate the probability to be the best option for every paywall variant participating in the test. This value is surely connected to the p-values we used before, but it is essentially a different approach, more robust, and easier to understand.
5. **Results interpretation:** Probability to be best is exactly how it sounds. The larger the probability is, the higher the likelihood of a specific option being the best choice for the task. You need to determine the threshold for decision-making yourself, it should depend on many other factors of your specific situation, but a common probability choice is 95%.
6. **Prediction intervals:** Adapty calculates prediction intervals for the performance metrics of each group, providing a range of values within which the true population parameter is likely to fall. This helps quantify the uncertainty associated with the estimated performance metrics.

Sample size determination

Determining an appropriate sample size is crucial for reliable and conclusive A/B test results. Adapty considers factors such as the statistical power and expected effect size, which continue to be important even under the Bayesian approach, to ensure an adequate sample size. The methods for estimating the required sample size, specific to the Bayesian approach we now employ, ensure the reliability of the analysis.

To learn more about the functionality of A/B tests, we recommend referring to our documentation on)creating and running A/B tests, as well as understanding the various A/B test metrics and results..

Adapty's analytical framework for A/B tests now employs a Bayesian approach, but the focus remains on the definition of metrics, formulation of hypotheses, and the selection of distributions. However, instead of determining p-values, we now compute the posterior distributions and calculate the probability of each variant being the best. We also determine the prediction intervals now. This revised approach, while still comprehensive and even more robust, is designed to provide insights that are more intuitive and easier to interpret. The goal remains to empower businesses to optimize their strategies, improve performance, and drive growth based on a robust statistical analysis of their A/B tests.

---

title: "Prediction in cohorts" description: ""

metadataTitle: ""

Adapty Predictions are designed to help you answer the following questions:

1. What is the predicted lifetime value (LTV) of your user cohorts?
2. Which cohorts are likely to generate the highest revenue in the future?
3. How much you can invest being aware of the predicted payoff?

With Adapty prediction, you can gain valuable insights and make data-driven decisions to optimize your revenue and growth strategy.

Adapty's prediction model is a powerful new feature that uses machine learning to help you to get a better understanding of the long-term revenue potential and behavior of your app's users. Using advanced gradient boosting techniques, the LTV prediction model can estimate the total revenue a user is expected to generate during the lifetime as a subscriber. This will help you to make informed decisions about user acquisition, marketing strategies, and product development.

Currently, the LTV prediction model provides two types of predictions: predicted LTV and predicted revenue.

Predictions in cohorts

Adapty now offers the ability to predict the lifetime value (LTV) of users and their predicted revenue for subscription-based apps. These predictions can be displayed on the cohort analysis page for 3, 6, and 12 months. However, it is important to note that the model does not currently work for lifetimes and for one-time purchases. Additionally, the accuracy of the LTV model may be lower for very new apps with limited data and for apps that have experienced significant changes in their user traffic.

Prediction model

The prediction model is built using gradient boosting, a highly accurate and efficient machine-learning algorithm that can handle large datasets. By leveraging transaction data from your subscription-based apps, our model can predict the LTV for users a year after their profile was created. The data used for the model is completely anonymized.

The model is trained on data from all of your apps within Adapty. However, the predicted values are further refined and tailored based on the specific behavior patterns observed in cohorts associated with each individual app. This post-correction algorithm ensures more accurate predictions, taking into account the unique characteristics of each app.

The model achieves a high level of accuracy, with a Mean Absolute Percentage Error (MAPE) of slightly below 10%. This level of precision allows businesses to confidently rely on the model's predictions when making data-driven decisions.

Adapty utilizes two distinct gradient-boosting models to forecast LTV:

- ***Revenue prediction model:*** Predicts the total revenue generated by a cohort of users.
- ***LTV prediction model:*** Predicts the average LTV of users in a cohort.

The trained models are then used to predict the total revenue and average LTV of each cohort within the selected period (3, 6, 9, or 12 months) after cohort creation. These predictions are verified and constrained to ensure they are consistent with typical user behavior patterns.

The predictions are initially generated after 1-3 weeks following the creation of the cohort. This timeframe allows for sufficient data gathering and analysis. Suppose a cohort is created on January 1st. Predictions for that cohort would be available sometime between January 15th and January 29th.

After the initial prediction generation, the predictions are then updated daily using the latest transactional data available for the cohort. This frequent updating ensures that the predictions remain current and reflect the most recent behavior of the cohort.

The model takes into account a variety of significant statistics pertaining to the cohorts, including revenue generated from past subscription periods, user retention rates, present LTV, subscription type, the proportion of users from Google Play and App Store, and geographic distribution of users by country, among others. These features are meticulously chosen to guarantee that the model captures the most pertinent information required to generate precise forecasts about the future LTV of users.

The model typically reflects changes in product performance, such as increased retention for monthly subscriptions, with a delay of approximately one week.

The ML model used to predict revenue and LTV has certain limitations that should be taken into account when interpreting its results. These limitations include:

- Data quality: The model's performance depends on the quality and representativeness of the data available. Data quality is a crucial aspect of data analysis, and one of the main reasons for insufficient results is when a cohort behaves unusually and deviates from normal cohort metrics for a particular app or all other apps. This type of data is considered uncommon for the model, which can lead to unexpected results. This situation is more common for new, unusual products or new apps that haven't been included in the training set.
- Time frame: The model can predict values up to 12 months from the creation of a user cohort. You will see growing actual revenue but the prediction will remain at the point of 12 months.
- Subscription durations: The model shows the best performance on monthly and weekly subscriptions

Prediction in Adapty cohorts

To access predicted revenue and predicted LTV values for your subscribers, you can navigate to the Cohort Analyses page in your Adapty dashboard. Also, if you want to learn more about the Adapty cohort, please reference our documentation about it.

./ width: '700px', / image width */display: 'block',* / for alignment */margin: '0 auto'* / center alignment */ }} />

The predicted revenue (pRevenue) column shows the estimated total revenue a cohort of subscribers is expected to generate during the selected time frame after cohort creation. This value is calculated using Adapty's revenue prediction model, which utilizes advanced gradient boosting techniques to predict revenue for users.

The predicted LTV (pLTV) column shows the estimated lifetime value of each user in the selected cohort. This value is calculated by dividing the predicted revenue by the predicted number of paying users in the cohort. The predicted number of paying users is calculated using Adapty's base prediction model, which predicts the number of paying users in a cohort.

To define the time period for which the predicted revenue and predicted LTV values are displayed, you can select the desired value from the timeframe dropdown in the user interface. The available options are typically 3, 6, 9, or 12 months after cohort creation.

To provide even more valuable insights, Adapty allows you to filter predicted revenue and LTV by product. By default, Adapty builds predictions based on all purchase data, but filtering by product can help you better understand how each product is performing and how it contributes to your overall predicted revenue and LTV.

./ width: '700px', / image width */display: 'block',* / for alignment */margin: '0 auto'* / center alignment */ }} />

When there is insufficient data available to generate accurate predictions, the corresponding fields will be greyed out in the user interface. Hovering over the greyed-out fields will display a tooltip with the message "**Insufficient data for accurate prediction**". This serves as a visual cue that the predicted values may not be reliable and further data collection and analysis may be necessary to generate accurate predictions. The lack of data may occur due to several reasons, such as insufficient time since cohort creation, a small cohort size, or an unpopular subscription type. In some cases, the cohort may behave unusually, deviating from the normal metrics used to train the model. Waiting a few weeks may help resolve this issue.

Also, another possible case is there are no values for the prediction. Empty results may occur if there is either insufficient data to make any predictions (usually at least 1-3 weeks of data is needed), or if the maximum time frame for prediction has already passed, i.e., it has been more than a year since the cohort was created.

:::warning When upgrading your pricing plan to the Pro+ or Enterprise, it's important to note that there may be a maximum delay of 24 hours before the prediction data for Revenue and LTV becomes available on your Adapty dashboard. :::

Adapty cohorts provides valuable insights into your revenue and user LTV (lifetime value) by showing current revenue and current money per paying user. By tracking these metrics, you can monitor how your actual numbers are progressing toward the predicted values and track your progress over time. While the primary purpose of using prediction numbers in building plans, it's important to exercise caution and not solely rely on these predictions for decision-making. Rather, they should be used as a guide to inform your strategy and help you make informed business decisions. By leveraging Adapty cohorts and predict, you can gain a better understanding of your revenue and user behavior, and use this information to optimize your business operations for greater success.

---

title: "Predictions in A/B tests" description: ""

metadataTitle: ""

Welcome to the Adapty Predictive Analytics documentation for our A/B testing feature. This tool will provide insights into the future results of your running A/B Tests and help you make data-driven decisions faster ðŸš€Â with Adapty's ML-powered predictions.

:::note A/B test winner predictions are only available on Pro+ and Enterprise plans :::

What are A/B test predictions?

Adapty's A/B Test Predictions employ advanced machine learning techniques (specifically gradient boosting models) to forecast the long-term revenue potential of the paywalls that are compared in an A/B test.

This predictive model enables you to select the most effective paywall based on projected revenue after a year, instead of relying only on the metrics you observe while the test is running. This allows you decide on the winner more reliably and faster, without having to wait weeks for the data to accumulate.

How does the model work?

The model is trained on extensive historical A/B test data from a variety of apps in different categories. It incorporates a wide range of features to predict the revenue a paywall is likely to generate in a year after the experiment start. These features include:

- User transactions and conversion rates over different periods
- Geographic distribution of users
- Platform usage (iOS or Android)
- Opt-out and refund rates
- Subscription products and their period lengths (daily, monthly, yearly and so on)
- Other transaction-related data

The model also accounts for trial periods in paywalls, using historical conversion rates to predict revenue as if users were already converted. This ensures a fair comparison between paywalls with and without trial offers, because we will also account for active trials potentially bringing in revenue in the future.

How is Predicted P2BB different from just the P2BB?

Our A/B tests utilise the Bayesian approach: basically we model the distribution of the revenue per user (or â€œRevenue per 1K usersâ€ to be specific) and then calculate the probability that one distribution is â€œtrulyâ€ better than the other one and not by a random chance â€" and this is what we call the Probability-to-be-the-best or P2BB (you can learn more about our approach here).

Itâ€™s important to note that while doing so, we only rely on the revenue that has been accumulated over the time the test has been running. So if you were to run a test comparing a yearly subscription to a weekly one, you would have to wait a really long time to truly understand what performs better. A similar thing happens when you compare trial subscriptions with non-trial subscriptions in an A/B test â€" as the active trials that could potentially shift the winner dynamics are always unaccounted for in the revenue.

This is where our predictive model comes into play. Having the current revenue distribution in an A/B test and trained on a large dataset itâ€™s able to predict the future version of the revenue distribution (namely after 1 year). And after doing so, it produces a predicted P2BB â€" the one that you would arrive at if you were to run the test for the entire year.

Note that sometimes predicted P2BB can contradict the current P2BB. When that's the case, we highlight the variation rows with yellow like so:

./2024-02-15at_13.08.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* / image width */display: 'block',* / for alignment */margin: '0 auto'* / center alignment */ }} />

We consider that a signal that you should accumulate more data to confirm the winner or dig deeper into the A/B test to find out the reason behind it. Generally we recommend trusting the predicted P2BB over the current P2BB because it simply takes more data into account, but the final decision is of course up to you.

Model accuracy and certainty

The model achieves a high level of accuracy, with a Mean Absolute Percentage Error (MAPE) of slightly below 10%. This level of precision allows businesses to confidently rely on the model's predictions when making data-driven decisions.

To further ensure stability, the model employs a 'certainty' criterion based on three factors:

- A narrow prediction interval - the model is confident in its outcome
- Sufficient amount of subscriptions & revenue in the test
- At least 2 weeks from the test start have passed

To assure the quality of the prediction is of the highest standards possible, prediction is considered reliable only if at least two of these criteria are met without completely failing the third.

When a new A/B test begins, the model provides a year-ahead revenue per 1k (our main A/B test metric) prediction for each paywall. Predictions are displayed only when they meet the certainty criteria. If the data is insufficient, the model will indicate "insufficient data for predictionâ€.

Limitations and considerations

While our predictive model is a powerful tool, it's important to consider its limitations.

The model's performance depends on the quality and representativeness of the available data. Unusual cohort behaviour or new apps not included in the training set can affect prediction accuracy.

Nevertheless, predictions are updated daily to reflect the latest data and user behaviors. This ensures that the insights you receive are always based on the most current information.

ðŸ§ Note: This tool is a supplement to, not a replacement for, your expert judgment and understanding of your app's unique dynamics. Use these predictions as a guide alongside other metrics and market knowledge to make informed decisions.

---

title: "Profiles/CRM" description: ""

metadataTitle: ""

Profiles is a CRM for your users. With Profiles, you can:

1. Find a user with any ID you have including email and phone number.
2. Explore the full payment path of a user including billing issues, grace periods, and other events.
3. Analyze user's properties such as subscription state, total revenue/proceeds, last seen, and more.
4. Grant the user a subscription.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

In a full table of subscribers, you can filter, sort, and find users. The state describes user state in terms of a subscription and can be:

| User state | Description | | :---------- | :---------- | | **Subscribed** | The user has an active subscription | | **Active trial** | The user has a subscription with an activ trial period | | **Auto renew off** | The user turned off auto-renewal. Check events for more info | | **Subscription cancelled** | The user cancelled a subscription. Check events for more info | | **Trial cancelled** | The user cancelled a trial | | **Never subscribed** | The user has never subscribed, i.e. he's a freemium user | | **Billing issue** | The user can't be charged | | **Grace period** | A user entered a grace period |

You can group users into Segment to create Promo Campaigns, group analysis, and more.

User properties

./2023-06-26at_20.32.232x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

You can send any properties that you want for the user.

By default, Adapty sets:

- **Customer user ID**. Is a developer ID and can be any string
- **Adapty ID**. Internal ID of a user in Adapty
- **IDFA**
- **Country**. From IP address country of the user
- **OS**
- **Device**
- **Created at**. Profile creation date
- **Last seen**

For a better understanding of your user, we suggest sending at least your internal user ID or user email. This will help you to find a user.

After installing SDK, Adapty automatically collects user events from the payment queue and displays them in a user profile.

Custom attributes

You can see custom attributes that were set either from SDK or manually assign them to the user using the Add attribute button in the Attributes section on the profile page.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Grant a subscription

In a profile, you can find an active subscription. At any time you can prolong the user's subscription or grant lifetime access.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

It's most useful for users without an active subscription so you can grant the individual user or a group of users premium features for some time. Please note that adjusting the subscription date for active subscriptions will not impact the ongoing payments.

:::note **Expires at** must be a date in the future and can't be decreased ones set. :::

Profile record creation

Adapty creates an internal profile ID for every user. However, if you have your own authentication system, set your own Customer User ID, a unique identifier for each user in your system. In this case. Adapty will add this ID to the user profile, which will give you several advantages:

1. All transactions and events will be tied to the same profile.
2. You can find users by their customer user ID in the **Profiles** section and view their transactions and events.
3. You can use the customer user ID in the server-side API.
4. The customer user ID will be sent to all integrations.

If no customer user ID is passed to Adapty, Adapty will create a new additional internal profile ID in the following cases:

- When a user launches your app for the first time after installation and reinstallation.
- When a user logs out of your app.

This means that a user who installs, then uninstalls, and reinstalls your app may have several profile records in Adapty if no customer user ID is used. All transactions in a chain are tied to the profile that generated the first transaction â€” the "original" profile. This helps keep a complete transaction history â€” including trial periods, subscription purchases, renewals, and more, linked to the same profile.

A new profile record that generates a subsequent transaction, called a "non-original" profile, may not have any events associated with it but will retain the granted access level. In some cases, you will also see "access*level*updated" events here.

Here is an example of a non-original profile. Notice the absence of events in the **User history** and the presence of an access level.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

---

title: "Segments" description: ""

metadataTitle: ""

A Segment is a group of users with common properties.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Segments are mainly used in Placements and in A/B tests to create an Audience and target it with a paywall (or multiple paywalls). Here are some example scenarios where that can be useful:

- targeting non-subscribed users with the default paywall and offering a discount for those who have previously canceled their subscription or a trial.
- having different paywalls for different countries
- basing your segment on the Apple Search Ads attribution data
- creating segments based on your app's version, so that once you introduce a new paywall that is only supported by the recent versions of your app, the older versions would still continue to work

Creation

To create a segment, write a segment name, and choose attributes.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

Available attributes

- Age of the user
- App User ID

- App Version
- Calculated subscription state
- Calculated total revenue USD
- Country from IP
- Country from store account
- Creation date
- Device
- Gender
- Language
- OS
- Platform
- Subscription expiration date
- Subscription product
- Attribution Source: Organic, Non-Organic, Unknown
- Attribution Channel
- Attribution Campaign
- Attribution Ad Group
- Attribution Ad Set
- Attribution Creative
- Custom Attributes

:::note Please note that these attributes are predefined and cannot be modified, except for the **App Version** attribute, which allows for adding new values. :::

Custom attributes

To create even more targeted segments, you can also create custom attributes. Custom attributes allow you to create user groups based on properties that are specific to your app or business.

:::note To create custom attributes, you can set them up in either the mobile SDK or the dashboard, and there is no specific order in which they need to be created. To set up custom attributes in the mobile SDK, please follow this link to learn how to set them up. :::

:::warning Adapty collects the **country of the store** for iOS devices with version 13 or higher. :::

To create custom attributes from the Adapty Dashboard, select the * *Create Custom Attributes** from the Select Attribute Dropdown options.

2023-03-16at_17.20.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Here's how to fill the fields for custom attributes. Also, you can read more about custom attribute validation rules here.

1. **Name** represents the name of the custom attribute and will be used in the Adapty dashboard only.
2. **Key** is the unique identifier for the custom attribute. This key value should match the key value used in the SDK.
3. **Type** field has two options. If you select "String", you have to enter a list of possible values for the attribute. If you select "Number", the attribute will accept only numeric values.
4. If you selected "String" as the type, enter a list of possible **values** for the attribute. If you selected "Number", the attribute will accept only numeric values.

Complete all the required fields first, and then you can begin utilizing the custom attribute in your segment definition. Once you've created your segments, you can use them to target A/B testing, among other things!

---

title: "Segments" description: ""

metadataTitle: ""

A Segment is a group of users with common properties.

./ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Segments are mainly used in Placements and in A/B tests to create an Audience and target it with a paywall (or multiple paywalls). Here are some example scenarios where that can be useful:

- targeting non-subscribed users with the default paywall and offering a discount for those who have previously canceled their subscription or a trial.
- having different paywalls for different countries
- basing your segment on the Apple Search Ads attribution data
- creating segments based on your app's version, so that once you introduce a new paywall that is only supported by the recent versions of your app, the older versions would still continue to work

Creation

To create a segment, write a segment name, and choose attributes.

./ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Available attributes

- Age of the user
- App User ID
- App Version
- Calculated subscription state
- Calculated total revenue USD
- Country from IP
- Country from store account
- Creation date
- Device
- Gender
- Language
- OS
- Platform
- Subscription expiration date
- Subscription product
- Attribution Source: Organic, Non-Organic, Unknown
- Attribution Channel
- Attribution Campaign
- Attribution Ad Group
- Attribution Ad Set
- Attribution Creative
- Custom Attributes

:::note Please note that these attributes are predefined and cannot be modified, except for the **App Version** attribute, which allows for adding new values. :::

Custom attributes

To create even more targeted segments, you can also create custom attributes. Custom attributes allow you to create user groups based on properties that are specific to your app or business.

:::note To create custom attributes, you can set them up in either the mobile SDK or the dashboard, and there is no specific order in which they need to be created. To set up custom attributes in the mobile SDK, please follow this link to learn how to set them up. :::

:::warning Adapty collects the **country of the store** for iOS devices with version 13 or higher. :::

To create custom attributes from the Adapty Dashboard, select the * *Create Custom Attributes** from the Select Attribute Dropdown options.

2023-03-16at_17.20.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Here's how to fill the fields for custom attributes. Also, you can read more about custom attribute validation rules here.

1. **Name** represents the name of the custom attribute and will be used in the Adapty dashboard only.
2. **Key** is the unique identifier for the custom attribute. This key value should match the key value used in the SDK.
3. **Type** field has two options. If you select "String", you have to enter a list of possible values for the attribute. If you select "Number", the attribute will accept only numeric values.
4. If you selected "String" as the type, enter a list of possible **values** for the attribute. If you selected "Number", the attribute will accept only numeric values.

Complete all the required fields first, and then you can begin utilizing the custom attribute in your segment definition. Once you've created your segments, you can use them to target A/B testing, among other things!

---

title: "Event feed" description: ""

metadataTitle: ""

Event feed allows you to visually track Events generated by Adapty and check the status of their export to 3rd-party integrations, including the webhook. Please note that transactions generated with server-side API do not appear in the **Event Feed**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

:::note AppsFlyer, Facebook Ads, and Branch sending status could be inaccurate because they do not always return errors when they occur. :::

To view the profile of the user who has initiated the transaction, click the **View Profile** button in the event details.

---

title: "Configure 3d-party integration" description: ""

metadataTitle: ""

With Adapty integrations, you can seamlessly transmit subscription events and purchase data to your preferred platform or workflow. Whether you're seeking user behavior insights, customer engagement strategies, or enhanced product analytics for your marketing team, Adapty can effortlessly forward in-app purchase events to your chosen integration.

Adapty effortlessly tracks in-app purchases and subscription events such as trials, conversions, renewals, and cancellations. These events are automatically communicated to your chosen integrations. This allows you to engage with customers based on their current stage and analyze revenue-related activities within your app.

Integration settings

2023-08-22at_13.26.562x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Each integration offers the following configuration options that impact all events sent through this integration:

| Setting | Description | | :---------- | :---------- | | **Reporting Proceeds** | Select how revenue values are presented: either net of App Store and Play Store commissio
gross (before deductions). Toggle the "Send sales as proceeds" checkbox to display sales as proceeds after App Store / Play Store commissions have been subtracted. | | **Send Trial Price** | If checked, Adapty will transmit the subscription price for the Trial Started event. | | **Exclude Historical Events** | Opt to exclude events that occurred before the user installed the app with Adapty SDK. This prevents duplication of events and ensures accurate reporting. For instance, if a user activated a monthly subscription on January 10th and updated the app with Adapty SDK on March 6th, Adapty will omit events before March 6th and retain subsequent events. | | **Report User's Currency** | Choose whether sales are reported in the user's currency or in USD. | | **Send User Attributes** | If you wish to send user-specific attributes, like language preferences, and your OneSignal plan supports more than 10 tags, select this option. Enabling this allows the inclusion of additional information beyond the default 10 tags. Note that exceeding tag limits may result in errors. | | **Send Attributions** | Enable this option to transmit attribution information (e.g. AppsFlyer attribution) and receive relevant details. |

Configure the events

Below the credentials, there are three groups of events you can send to the selected integration platform from Adapty. You should turn on the ones you need.

It's important to note that event name customization is available for certain integrations, while for others, the event names are set and cannot be modified. Additionally, with certain integrations such as Airbridge for example, you have the flexibility to associate multiple event names with a single Adapty event. Check the full list of the Events offered by Adapty here.

3107.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

While we recommend utilizing Adapty's default event names, you have the freedom to adapt event names as per your specific requirements.

---

title: "Events to send to 3d-party integrations" description: ""

metadataTitle: ""

Apple and Google send subscription events directly to the server using App Store Server Notifications and Real-time Developer Notifications (RTDN). Therefore, mobile apps cannot send events to analytical systems correctly and on-time. For example, if the user subscribed and then didn't open the app, without a server developer will get zero information about subscription status.

After installing Adapty SDK and setting up App Store Server Notifications for iOS and Real-time Developer Notifications (RTDN). for Android, Adapty receives info about your customer behavior and converts it into human-readable events.

:::note Adapty processes events into human-readable format instantly as they created and enriches them with additional information, such as customer ID, consecutive payments, store commission info, and others. Besides that Apple doesn't send events about subscription renewals but we do. :::

Events

| Event Name | Description | | :---------- | :---------- | | **subscription*started*** | A user has activated a subscription without a trial period i.e. he was billed instantly. | | **subscriptionrenewed** | A subscription was renewed and the user was charged. For both trial and non-trial subscriptions, this event is sent starting from the second billing. | | **subscription*expired*** | A user has canceled a subscription and it is completely finished. | | **trialstarted** | A user has activated a trial subscription. | | **trial*converted*** | A trial period has ended and the user was billed, i.e. first purchase was made. | | **trialexpired** | A trial has expired without converting to a subscription. | | **non*subscription*purchase** | Any non-subscription purchase e.g. lifetime access or consumable product such as coins. | | **billing*issue*detected** | An attempt to charge the user was made, but a billing issue happened. Usually, it means the user doesn't have enough card balance. | | **entered*grace*period** | The payment was not successful and the user entered into a grace period. The user still has access to the premium features of your app until the grace period is finished. | | **trial*renewal*cancelled** | A user turned off subscription auto-renewal during the trial. A user still has access to the premium features of your app until the end of the trial period. | | **trial*renewal*reactivated** | A user turned on subscription auto-renewal during the trial period. | | **subscription*renewal*cancelled** | A user turned off subscription auto-renewal. A user still has access to the premium features of your app until the end of the subscription period. | | **subscription*renewal*reactivated** | A user turned on subscription auto-renewal. | | **subscription*refunded*** | A subscription was refunded (e.g. by Apple support). | | **nonsubscription*purchase*refunded** | Non-subscription purchase was refunded. | | **subscription*paused*** | User activated subscription pause (Android only). | | **subscriptiondeferred** | A user's subscription has been deferred, ie they were granted free usage time (Android only). Usually, it happens in response to an API call from your servers. | | **access*level*updated** | User's access level updated (Webhook only). |

:::note **SUBSCRIPTION_EXPIRED**(previously **SUBSCRIPTION_CANCELED**) event means that the subscription completely finished and the user has no longer access to the premium features of the app. When the user unsubscribes, **AUTO_RENEW_OFF** or **AUTO_RENEW_OFF_SUBSCRIPTION** is sent. The same logic applied to **TRIAL_CANCELLED.** :::

The events above fully cover the users' state in terms of purchases. Let's look at some examples.

Example 1

The user has activated a monthly subscription on April 1st with 7 days trial. On the 4th day, he unsubscribed.

In that case following events will be sent:

1. trial_started on April 1st
2. auto_renew_off on 4th April
3. trial_cancelled on 7th April

Example 2

The user has activated a monthly subscription on April 1st with 7 days trial. On the 10th day, he unsubscribed.

In that case following events will be sent:

1. trial_started on April 1st
2. trial_converted on April 7th
3. auto_renew_off_subscription on April 10th
4. subscription_cancelled on May 1st

Properties

| Property | Type | Description | |--------|----|-----------| **profile*id*** | uuid | Adapty user ID. | | **currency** | str | Local currency (defaults to USD). | | **priceusd** | float | Product price before Apple/Google cut. Revenue. | | **proceeds*usd*** | float | Product price after Apple/Google cut. Net revenue. | | **netrevenue*usd*** | float | Net revenue (income after Apple/Google cut and taxes) in USD. Can be empty. | | **pricelocal** | float | Product price before Apple/Google cut in local currency. Revenue. | | **proceeds*local*** | float | Product price after Apple/Google cut in local currency. Net revenue. | | **transactionid** | str | A unique identifier for a transaction such as a purchase or renewal. | | **original*transaction*id** | str | The transaction identifier of the original purchase. | | **purchase*date*** | ISO 8601 date | The date and time of product purchase. | | **originalpurchase*date*** | ISO 8601 date | The date and time of the original purchase. | | **environment** | str | Can be _Sandbox or *Production*. | | **vendor*product*id** | str | Product ID in the Apple App Store, Google Play Store, or Stripe. | | **base*plan*id** | str | Base plan ID in the Google Play Store or price ID in Stripe. | | **event*datetime*** | ISO 8601 date | The date and time of the event. | | **store** | str | Can be _appstore_ or *play*store. | | ***trial***duration | str | Duration of a trial period in days. Sent in a

format "{} days" , for example, "7 days". | | **cancellation*reason*** | str |

A reason why the user canceled a subscription.

Can be

iOS & Android

voluntarily*cancelled*, *billing*error, *_refund*

iOS

*price*increase, *_product*was*not*available, *_unknown*

Android

*new*subscription*replace*, *cancelled*by*developer*

| | **subscription*expires*at** | ISO 8601 date | The Expiration date of subscription. Usually in the future. | | **consecutive*payments*** | int | The number of periods, that a user is subscribed to without interruptions. Includes the current period. | | **rateafter*first*year** | bool | Boolean indicates that a vendor reduces cuts to 15%. Apple and Google have 30% first-year cut and 15% after it. | |**promotional*offer*id** | str | ID of promotional offer as indicated in the Product section of the Adapty Dashboard | | **store*offer*category** | str | Can be *introductory* or *promotional*. | | **store*offer*discount*type*** | str | Can be _freetrial, *_pay*as*you*go_* or *pay*up*front*. | | **paywall*name*** | str | Name of the paywall where the transaction originated. | | **paywallrevision** | int | Revision of the paywall where the transaction originated. The value is set to 1. | | **developer*id*** | str | Developer (SDK) ID of the placement where the transaction originated. | | **abtest*name*** | str | Name of the A/B test where the transaction originated. | | **abtest*revision*** | int | Revision of the A/B test where the transaction originated. The value is set to 1. | | **cohortname** | str | Name of the audience to which the profile belongs to. | | **profile*event*id** | uuid | Unique event ID that can be used for deduplication. | | **store*country*** | str | The country sent to us by the store. | | **profileip*address*** | str | Profile IP (can be IPv4 or IPv6, with IPv4 preferred when available). It is updated each time IP of the device changes. | | **profilecountry** | str | Determined by Adapty, based on profile IP. | | **profile*total*revenue*usd*** | float | Total revenue for the profile, refunds included. | | **variationid** | uuid | Unique ID of the paywall where the purchase was made. | | **access*level*id** | str | Paid access level ID | | **is*active*** | bool | Boolean indicating whether paid access level is active for the profile. | | **willrenew** | bool | Boolean indicating whether paid access level will be renewed. | | **is*refund*** | bool | Boolean indicating whether transaction is refunded. | | **islifetime** | bool | Boolean indicating whether paid access level is lifetime. | | **is*in*grace*period*** | bool | Boolean indicating whether profile is in grace period. | | **startsat** | ISO 8601 date | Date and time when paid access level starts for the user. | | **renewed*at*** | ISO 8601 date | Date and time when paid access will be renewed. | | **expiresat** | ISO 8601 date | Date and time when paid access will expire. | | **activated*at*** | ISO 8601 date | Date and time when paid access was activated. | | **billingissue*detected*at** | ISO 8601 date | Date and time of billing issue. |

Each event has the following properties:

`transaction_id, original_transaction_id, purchase_date, original_purchase_date, environment, vendor_product_id, event_datetime, store.`

In addition, some events have additional properties. For the events `subscription_refunded` and `non_subscription_purchase_refunded`, it is mandatory to provide the values of `price_usd` and `proceeds_usd` as additional properties.

| Event Name | Properties | | :---------- | :---------- | | **subscription_initial_purchase** | price_usd, proceeds_usd, subscription_expires_at, consecutive_payments, rate_after_first_year, trial_duration | | **subscription_renewed** | price_usd, proceeds_usd, subscription_expires_at, consecutive_payments, rate_after_first_year, trial_duration | | **subscription_cancelled** | cancellation_reason, trial_duration | | **trial_started** | subscription_expires_at, trial_duration | | **trial_converted** | price_usd, proceeds_usd, subscription_expires_at, consecutive_payments, rate_after_first_year, trial_duration | | **trial_cancelled** | cancellation_reason, trial_duration | | **non_subscription_purchase** | price_usd, proceeds_usd | | **billing_issue_detected** | subscription_expires_at, trial_duration | | **entered_grace_period** | subscription_expires_at, trial_duration |

Event example

```
json title="Json" { "price_usd": 9.99, "proceeds_usd": 6.99, "transaction_id": "1000000628581600", "original_transaction_id": "1000000628581600", "purchase_date": "2020-02-18T18:40:22.000000+0000", "original_purchase_date": "2020-02-18T18:40:22.000000+0000", "environment": "Sandbox", "vendor_product_id": "premium", "event_datetime": "2020-02-18T18:40:22.000000+0000", "store": "app_store" }
```

Adapty sends events to your server and 3rd party analytical systems.

profile*ip*address property is synchronized with the current device IP. Each time the Adapty servers receive info from the SDK, the IP will be updated if it differs from the one we have on record.

Access level updated

Adapty has a special event `access_level_updated`. It is sent only to webhook integration every time the access level is updated/set for a specific customer. Use this event to update a customer's subscription in your database/system. Before you had to keep track of several events to sync subscription state and some cases were not covered, like setting access level manually from Adapty CRM. Now, no matter what was the source of access level changes, you will always receive a dedicated event for that, therefore it's more precise and has more details than `subscription_renewed`, `trial_started`, `entered_grace_period`, etc.

2022-05-03at_14.22.56.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

| Property | Type | | :---------- | :---------- | | **store** | str | | **currency** | str | | **is*active*** | bool | | **isrefund** | bool | | **expires*at*** | ISO 8601 date | | **starts at** | ISO 8601 date | | **profileid** | str | | **renewed*at*** | ISO 8601 date | | **willrenew** | bool | | **environment** | str | | **is*lifetime*** | bool | | **activatedat** | ISO 8601 date | | **purchase*date*** | ISO 8601 date | | **storecountry** | str | | **event*datetime*** | ISO 8601 date | | **transactionid** | str | | **access*level*id** | str | | **profile*country*** | str | | **profileevent*id*** | str | | **vendorproduct*id*** | str | | **is*in*grace*period*** | bool | | **original*purchase*date** | ISO 8601 date | | **original*transaction*id** | str | | **subscription*expires*at** | ISO 8601 date | | **profile*total*revenue*usd*** | float | | **billingissue*detected*at** | ISO 8601 date | | **cancellation*reason*** | str | | **activeintroductory*offer*type** | str | | **active*promotional*offer*type*** | str | | **activepromotional*offer*id** | str |

We don't send `access_level_updated` upon subscription expiration - please, refer to **expires_at** value to end the subscriptions on your side.

Please note that some properties can only be set using the grant Server-Side API method. You can find more information about this method in the Adapty Server-Side API documentation.

For detailed descriptions of the mentioned properties, you can refer to the API objects documentation.

Sending failed

We determine the deliverability based on HTTP status and consider everything **outside the 200-399 range** to be a fail.

You can see the status of certain integration events in the event list in the Adapty Dashboard. The system displays the statuses of integrations that are enabled for the app, regardless of whether the event type is enabled or disabled for a specific integration. If the event type is disabled for a particular integration, it will be color-coded as gray in the event feed. If there are any issues with integration, the integration name will be highlighted in red, indicating that attention is required to resolve the problem. In addition, the system provides tooltips when you hover over the integration name. These tooltips offer more detailed information about the reasons for the non-delivery of an event.

2023-06-02at_14.58.48.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

The Event feed utilizes a limitation on the number of events displayed, showing data from the past two weeks. This implementation enhances the loading speed of the page, enabling users to navigate and analyze data more efficiently.

---

title: "Attribution integration" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty allows easy integration with the popular attribution services: AppsFlyer, Adjust, Branch, Apple Search Ads, and Facebook Ads. Adapty will send subscription events to these services so you can accurately measure the performance of ad campaigns. You can also filter charts data using attribution data.

Send subscription events with correct user properties and ID's to attributions services you use.

Follow our detailed guidance on configuring the following 3d-part attribution integrations:

- Adjust
- Airbridge
- Apple Search Ads
- AppsFlyer
- Asapty
- Branch
- Facebook Ads
- Singular

:::note Don't see your attribution provider?

Let us know! Write to the Adapty support and we'll consider adding it. :::

Important

:::warning Avoiding events duplication

Make sure to turn off sending subscription events from devices and your server to avoid duplication.

If you set up direct integration with Facebook, turn off events forwarding from AppsFlyer, Adjust, or Branch. :::

:::info Be sure you've set up attribution integration in Adapty Dashboard, otherwise, we won't be able to send subscription events. :::

:::note Attribution data is set for every profile one time, we won't override the data once it's been saved. :::

Setting attribution data

To set attribution data for the profile, use `.updateAttribution()` method:

```
swift Adapty.updateAttribution("<attribution>", source: "<source>", networkUserId: "<networkUserId>") { error in if error == nil { // succesfull attribution update } } kotlin
Adapty.updateAttribution("<attribution>", "<source>", "<networkUserId>") { error -> if (error == null) { // succesfull attribution update } } java Adapty.updateAttribution("
<attribution>", "<source>", "<networkUserId>", error -> { if (error == null) { // succesfull attribution update } }); javascript try { await Adapty().updateAttribution("
<attribution>", source: "<source>", networkUserId: "<networkUserId>"); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } ```csharp Adapty.UpdateAttribution("", source, "", (error) => { if (error != null) { // handle the error }

    // succesfull attribution update

}); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript // Optionally import enum to JavaScript import { AttributionSource } from 'react-native-adapty';

const attribution = { /* ... */ }; try { await adapty.updateAttribution( attribution, AttributionSource.Branch, // or just 'branch' 'networkUserId' ); // succesfull attribution update } catch (error) { // handle AdaptyError } ```
```

Request parameteres:

- **Attribution** (required): a dictionary containing attribution (conversion) data.
- **Source** (required): a source of attribution. The allowed values are:
    - `.appsflyer`
    - `.adjust`
    - `.branch`
    - `.custom`
- **Network user Id** (optional): a string profile's identifier from the attribution service.

AppsFlyer

:::warning iOS SDK

To set attribution from AppsFlyer, pass the attribution you receive from the delegate method of AppsFlyer iOS SDK. Don't forget to set `networkUserId`. You should also configure AppsFlyer integration in Adapty Dashboard. :::

:::warning In this case, it is mandatory to pass the `networkUserId` parameter. :::

```
swift // Find your implementation of AppsFlyerLibDelegate // and update onConversionDataSuccess method: func onConversionDataSuccess(_ installData: [AnyHashable : Any]) { // It's
important to include the network user ID Adapty.updateAttribution(installData, source: .appsflyer, networkUserId: AppsFlyerLib.shared().getAppsFlyerUID()) } kotlin val
conversionListener: AppsFlyerConversionListener = object : AppsFlyerConversionListener { override fun onConversionDataSuccess(conversionData: Map<String, Any>) { // It's important
to include the network user ID Adapty.updateAttribution( conversionData, AdaptyAttributionSource.APPSFLYER, AppsFlyerLib.getInstance().getAppsFlyerUID(context) ) { error -> if
(error != null) { //handle error } } } } java AppsFlyerConversionListener conversionListener = new AppsFlyerConversionListener() { @Override public void
onConversionDataSuccess(Map<String, Object> conversionData) { // It's important to include the network user ID Adapty.updateAttribution( conversionData,
AdaptyAttributionSource.APPSFLYER, AppsFlyerLib.getInstance().getAppsFlyerUID(context), error -> { if (error != null) { //handle error } } ); } }; ```javascript @override Future initialize()
async { appsflyerSdk.onInstallConversionData((data) { try { await Adapty().updateAttribution(data, source: AdaptyAttributionSource.appsflyer, networkUserId: await appsflyerSdk.getAppsFlyerUID()); } on AdaptyError
catch (adaptyError) { // handle the error } catch (e) { }
});

await appsflyerSdk.initSdk(
    registerConversionDataCallback: true,
    registerOnAppOpenAttributionCallback: true,
    registerOnDeepLinkingCallback: true
);

return Future<bool>.value(true);

} ```
```

Adjust

:::warning iOS SDK

To set attribution from Adjust, pass the attribution you receive from the delegate method of Adjust iOS SDK. You should also configure Adjust integration in Adapty Dashboard. :::

```
swift // Find your implementation of AdjustDelegate // and update adjustAttributionChanged method: func adjustAttributionChanged(_ attribution: ADJAttribution?) { if let attribution
= attribution?.dictionary() { Adapty.updateAttribution(attribution, source: .adjust) } } kotlin adjustConfig.setOnAttributionChangedListener { attribution -> attribution?.let {
attribution -> Adapty.updateAttribution(attribution, AdaptyAttributionSource.ADJUST) { error -> if (error != null) { //handle error } } } } java
adjustConfig.setOnAttributionChangedListener(attribution -> { if (attribution != null) { Adapty.updateAttribution(attribution, AdaptyAttributionSource.ADJUST, error -> { if (error
!= null) { //handle error } }); } });
```

Branch

:::warning iOS SDK

To connect Branch user and Adapty user, make sure you set your `customerUserId` as Branch Identity Id. If you prefer to not use `customerUserId` in Branch, set `networkUserId` param in `.updateAttribution()` method to specify the Branch user Id. :::

```
swift // Pass the attribution you receive from the initializing method of Branch iOS SDK to Adapty. Branch.getInstance().initSession(launchOptions: launchOptions) { (data, error) in
if let data = data { Adapty.updateAttribution(data, source: .branch) } } kotlin object branchListener : Branch.BranchReferralInitListener { override fun
onInitFinished(referringParams: JSONObject?, error: BranchError?) { referringParams?.let { data -> Adapty.updateAttribution(data, AdaptyAttributionSource.BRANCH) { error -> if
(error != null) { //handle error } } } } } java Branch.BranchReferralInitListener branchListener = (data, e) -> { if (data != null) { Adapty.updateAttribution(data,
AdaptyAttributionSource.BRANCH, error -> { if (error != null) { //handle error } }); } };
```

:::note You should also configure Branch integration in Adapty Dashboard. :::

Apple Search Ads

Adapty can automatically collect Apple Search Ad attribution data. All you need is to add `AdaptyAppleSearchAdsAttributionCollectionEnabled` to the appâ€™s `Info.plist` file and set it to `YES` (boolean value).

Facebook Ads

Because of iOS IDFA changes in iOS 14.5, if you use Facebook integration, make sure you send `facebookAnonymousId` to Adapty via `.updateProfile()` method. It allows Facebook to handle events if IDFA is not available. You should also configure Facebook Ads in Adapty Dashboard.

```swift let builder = ProfileParameterBuilder() .with(facebookAnonymousId: FBSDKCoreKit.AppEvents.anonymousID)

Adapty.updateProfile(parameters: builder.build()) { error in if error == nil { // successful update } } </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin val builder = AdaptyProfileParameters.Builder() .withFacebookAnonymousId(AppEventsLogger.getAnonymousAppDeviceGUID(context))

Adapty.updateProfile(builder.build()) { error -> if (error == null) { // successful update } } </TabItem> <TabItem value="java" label="Java" default> java AdaptyProfileParameters.Builder builder = new AdaptyProfileParameters.Builder() .withFacebookAnonymousId(AppEventsLogger.getAnonymousAppDeviceGUID(context));

Adapty.updateProfile(builder.build(), error -> { if (error == null) { // successful update } }); ```
```

Custom

If you use another attribution system, you can pass the attribution data to Adapty. You can then segment users based on this data.
To set attributes, use only the keys from the example below (all keys are optional). The system supports max 30 available attributes, where the keys are limited to 30 characters. Every value in the map should be no longer than 50 characters. `status` can only be `organic`, `non-organic` or `unknown`. Any additional keys will be omitted.

```
swift title="Swift" let attribution = [ "status": "non_organic|organic|unknown", "channel": "Google Ads", "campaign": "Christmas Sale", "ad_group": "ad group", "ad_set": "ad set",
"creative": "creative id" ] Adapty.updateAttribution(attribution, source: .custom)
```

title: "Adjust" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adjust is one of the leading Mobile Measurement Partner (MMP) platforms, that collects and presents data from marketing campaigns. This helps companies see track their campaign performance.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in Adjust and analyze precisely how much revenue your campaigns generate.

The integration between Adapty and Adjust works in two main ways.

1. **Receiving attribution data from Adjust**
   Once you've set up the Adjust integration, Adapty will start receiving attribution data from Adjust. You can easily access and view this data on the user's profile page.

2023-08-11at_14.39.182x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

2. **Sending subscription events to Adjust**
   Adapty can send all subscription events which are configured in your integration to Adjust. As a result, you'll be able to track these events within the Adjust dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

How to set up Adjust integration

To setup the integration with Adjust go to Integrations > Adjust in the Adapty Dashboard, turn on a toggle from off to on, and fill out fields.

The next step of the integration is to set credentials.

2023-08-11at_14.43.382x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

1. If you have enabled OAuth authorization on the Adjust platform, it is mandatory to provide an *OAuth Token *during the integration process for your iOS and Android apps.
2. Next, you need to provide the **app tokens** for your iOS and Android apps. Open your Adjust dashboard and you'll see your apps.

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

:::note You may have different Adjust applications for iOS and Android, so in Adapty you have two independent sections for that. If you have only one Adjust app, just fill in the same information :::

You will need to copy **App Token** and paste it to Adapty.

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Another important thing is that Adjust doesn't support events older than 58 days. So, if you have an event that is more than 58 days old, Adapty will send it to Adjust, but the event datetime will be replaced by the current timestamp.

Events and tags

Adjust works a bit differently from other platforms. You need to manually create events in Adjust dashboard, get event tokens, and copy-paste them to appropriate events in Adapty.

So first step here is to find event tokens for all events that you want Adapty to send. To do that go to All Settings in your Adjust dashboard.

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Copy the event token and paste it to Adapty. Below the credentials, there are three groups of events you can send to Adjsut from Adapty. Check the full list of the events offered by Adapty here.

2023-08-11at_14.55.222x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Adapty will send subscription events to Adjust using a server-to-server integration, allowing you to view all subscription events in your Adjust dashboard and link them to your acquisition campaigns.

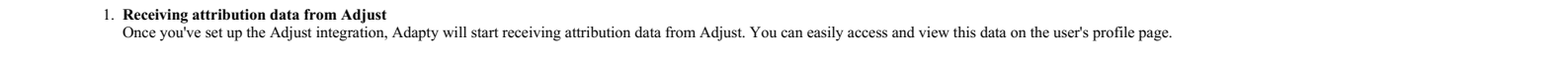
SDK configuration

It's very important to send Adjust attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

```
swift // Find your implementation of AdjustDelegate // and update adjustAttributionChanged method: func adjustAttributionChanged(_ attribution: ADJAttribution?) { if let attribution = attribution?.dictionary() { Adapty.updateAttribution(attribution, source: .adjust) } } kotlin val config = AdjustConfig(context, adjustAppToken, environment) config.setOnAttributionChangedListener { attribution -> attribution?.let { attribution -> Adapty.updateAttribution(attribution, AdaptyAttributionSource.ADJUST) { error -> if (error != null) { //handle error } } } } Adjust.onCreate(config) ```javascript import 'package:adjust*sdk/adjust.dart'; import 'package:adjust*sdk/adjust_config.dart';

AdjustConfig config = new AdjustConfig('{YourAppToken}', AdjustEnvironment.sandbox); config.attributionCallback = (data) async { var attribution = Map(); if (data.trackerToken != null) attribution['trackerToken'] = data.trackerToken!; if (data.trackerName != null) attribution['trackerName'] = data.trackerName!; if (data.network != null) attribution['network'] = data.network!; if (data.adgroup != null) attribution['adgroup'] = data.adgroup!; if (data.creative != null) attribution['creative'] = data.creative!; if (data.clickLabel != null) attribution['clickLabel'] = data.clickLabel!; if (data.adid != null) attribution['adid'] = data.adid!; if (data.costType != null) attribution['costType'] = data.costType!; if (data.costAmount != null) attribution['costAmount'] = data.costAmount!.toString(); if (data.costCurrency != null) attribution['costCurrency'] = data.costCurrency!; if (data.fbInstallReferrer != null) attribution['fbInstallReferrer'] = data.fbInstallReferrer!;

    try {
      await Adapty().updateAttribution(attribution, source: AdaptyAttributionSource.adjust);
    } on AdaptyError catch (adaptyError) {
      // handle error
    } catch (e) {}
  };

</TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp AdjustConfig adjustConfig = new AdjustConfig("{Your App Token}", AdjustEnvironment.Sandbox);
adjustConfig.setAttributionChangedDelegate(this.attributionChangedDelegate);

public void attributionChangedDelegate(AdjustAttribution attribution) { Dictionary data = new Dictionary();

if (attribution.adid != null) data["adid"] = attribution.adid;
if (attribution.network != null) data["network"] = attribution.network;
if (attribution.campaign != null) data["campaign"] = attribution.campaign;
if (attribution.adgroup != null) data["adgroup"] = attribution.adgroup;
if (attribution.creative != null) data["creative"] = attribution.creative;

String attributionString = JsonUtility.ToJson(data);
Adapty.UpdateAttribution(attributionString, AttributionSource.Adjust, (error) => {
    // handle the error
});

} </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { Adjust, AdjustConfig } from "react-native-adjust"; import { adapty } from "react-native-adapty";

var adjustConfig = new AdjustConfig(appToken, environment);

// Before submiting Adjust config... adjustConfig.setAttributionCallbackListener(attribution => { // Make sure Adapty SDK is activated at this point // You may want to lock this thread awaiting of `activate`
adapty.updateAttribution(attribution, "adjust"); });

// ... Adjust.create(adjustConfig); ```

---

title: "Adjust" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adjust is one of the leading Mobile Measurement Partner (MMP) platforms, that collects and presents data from marketing campaigns. This helps companies see track their campaign performance.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in Adjust and analyze precisely how much revenue your campaigns generate.
```

The integration between Adapty and Adjust works in two main ways.

1. **Receiving attribution data from Adjust**
   Once you've set up the Adjust integration, Adapty will start receiving attribution data from Adjust. You can easily access and view this data on the user's profile page.

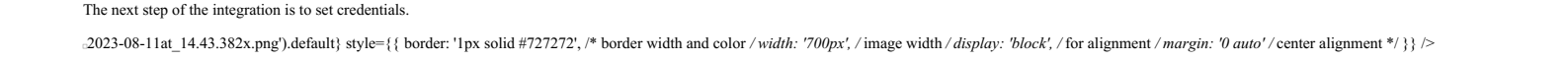


2. **Sending subscription events to Adjust**
   Adapty can send all subscription events which are configured in your integration to Adjust. As a result, you'll be able to track these events within the Adjust dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

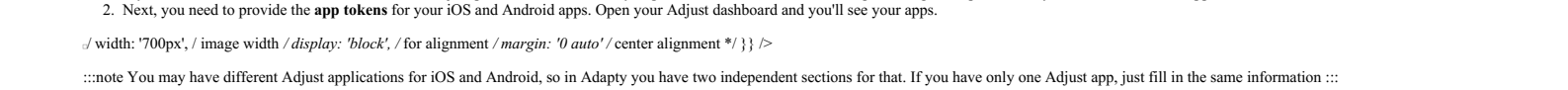
How to set up Adjust integration

To setup the integration with Adjust go to Integrations > Adjust in the Adapty Dashboard, turn on a toggle from off to on, and fill out fields.

The next step of the integration is to set credentials.

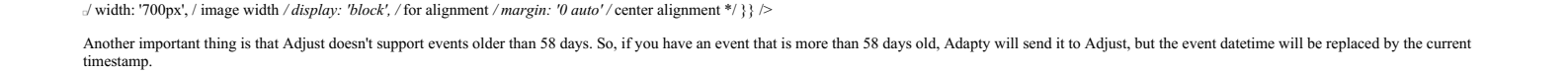


1. If you have enabled OAuth authorization on the Adjust platform, it is mandatory to provide an *OAuth Token *during the integration process for your iOS and Android apps.
2. Next, you need to provide the **app tokens** for your iOS and Android apps. Open your Adjust dashboard and you'll see your apps.



:::note You may have different Adjust applications for iOS and Android, so in Adapty you have two independent sections for that. If you have only one Adjust app, just fill in the same information :::

You will need to copy **App Token** and paste it to Adapty.

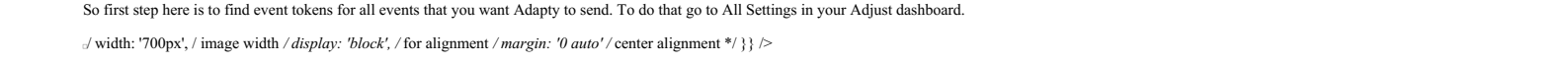


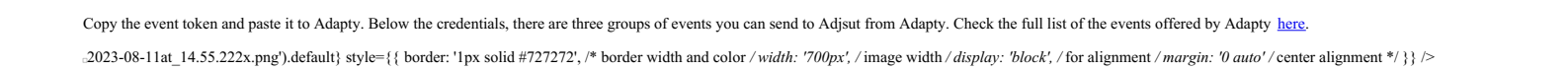
Another important thing is that Adjust doesn't support events older than 58 days. So, if you have an event that is more than 58 days old, Adapty will send it to Adjust, but the event datetime will be replaced by the current timestamp.

Events and tags

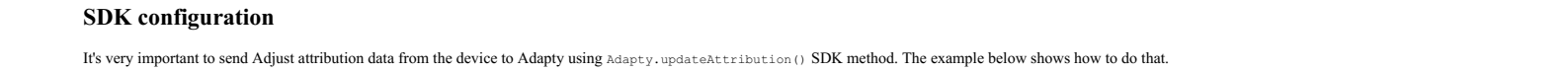
Adjust works a bit differently from other platforms. You need to manually create events in Adjust dashboard, get event tokens, and copy-paste them to appropriate events in Adapty.

So first step here is to find event tokens for all events that you want Adapty to send. To do that go to All Settings in your Adjust dashboard.





Copy the event token and paste it to Adapty. Below the credentials, there are three groups of events you can send to Adjsut from Adapty. Check the full list of the events offered by Adapty here.



Adapty will send subscription events to Adjust using a server-to-server integration, allowing you to view all subscription events in your Adjust dashboard and link them to your acquisition campaigns.

SDK configuration

It's very important to send Adjust attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

```
swift // Find your implementation of AdjustDelegate // and update adjustAttributionChanged method: func adjustAttributionChanged(_ attribution: ADJAttribution?) { if let attribution = attribution?.dictionary() { Adapty.updateAttribution(attribution, source: .adjust) } } kotlin val config = AdjustConfig(context, adjustAppToken, environment) config.setOnAttributionChangedListener { attribution -> attribution?.let { attribution -> Adapty.updateAttribution(attribution, AdaptyAttributionSource.ADJUST) { error -> if (error != null) { //handle error } } } } Adjust.onCreate(config) ```javascript import 'package:adjust*sdk/adjust.dart'; import 'package:adjust*sdk/adjust_config.dart';
```

AdjustConfig config = new AdjustConfig('{YourAppToken}', AdjustEnvironment.sandbox); config.attributionCallback = (data) async { var attribution = Map(); if (data.trackerToken != null) attribution['trackerToken'] = data.trackerToken!; if (data.trackerName != null) attribution['trackerName'] = data.trackerName!; if (data.network != null) attribution['network'] = data.network!; if (data.adgroup != null) attribution['adgroup'] = data.adgroup!; if (data.creative != null) attribution['creative'] = data.creative!; if (data.clickLabel != null) attribution['clickLabel'] = data.clickLabel!; if (data.adid != null) attribution['adid'] = data.adid!; if (data.costType != null) attribution['costType'] = data.costType!; if (data.costAmount != null) attribution['costAmount'] = data.costAmount!.toString(); if (data.costCurrency != null) attribution['costCurrency'] = data.costCurrency!; if (data.fbInstallReferrer != null) attribution['fbInstallReferrer'] = data.fbInstallReferrer!;

```
    try {
      await Adapty().updateAttribution(attribution, source: AdaptyAttributionSource.adjust);
    } on AdaptyError catch (adaptyError) {
      // handle error
    } catch (e) {}
  };
```

```
</TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp AdjustConfig adjustConfig = new AdjustConfig("{Your App Token}", AdjustEnvironment.Sandbox);
```

adjustConfig.setAttributionChangedDelegate(this.attributionChangedDelegate);

public void attributionChangedDelegate(AdjustAttribution attribution) { Dictionary data = new Dictionary();

```
if (attribution.adid != null) data["adid"] = attribution.adid;
if (attribution.network != null) data["network"] = attribution.network;
if (attribution.campaign != null) data["campaign"] = attribution.campaign;
if (attribution.adgroup != null) data["adgroup"] = attribution.adgroup;
if (attribution.creative != null) data["creative"] = attribution.creative;

String attributionString = JsonUtility.ToJson(data);
Adapty.UpdateAttribution(attributionString, AttributionSource.Adjust, (error) => {
    // handle the error
});
```

```
} </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { Adjust, AdjustConfig } from "react-native-adjust"; import { adapty } from "react-native-adapty";
```

var adjustConfig = new AdjustConfig(appToken, environment);

// Before submiting Adjust config... adjustConfig.setAttributionCallbackListener(attribution => { // Make sure Adapty SDK is activated at this point // You may want to lock this thread awaiting of `activate` adapty.updateAttribution(attribution, "adjust"); });

// ... Adjust.create(adjustConfig); ```

---

title: "Airbridge" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Airbridge offers an integrated marketing performance analysis for websites and mobile apps by consolidating data collected from multiple devices, platforms, and channels. Using Airbridge's Identity Resolution Engine, you can combine scattered customer identity data from web and app interactions into a unified people-based identity, resulting in more accurate attribution.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

The integration between Adapty and Airbridge operates in two main ways.

1. **Receiving attribution data from Airbridge**
   Once you've set up the Airbridge integration, Adapty will start receiving attribution data from Airbridge. You can easily access and view this data on the user's page.
2. **Sending subscription events to Airbridge**
   Adapty can send all subscription events which are configured in your integration to Airbridge. As a result, you'll be able to track these events within the Airbridge dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

How to set up Airbridge integration

To integrate Airbridge go to Integrations > Airbridge, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between your Airbridge and Adapty profiles. Airbridge app name and Airbridge API token are required.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

They both can be found in your Airbridge dashboard in the Third-party Integrations > Adapty section.

./2023-02-21at*11.19.29*AM.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Adapty API token field is pre-generated on the Adapty backend. You will need to copy the value of Adapty API token and paste it into the Airbridge Dashboard in the Adapty Authorization Token field.

./2023-03-01at_17.11.412x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Events and tags

Below the credentials, there are three groups of events you can send to Airbridge from Adapty

./2023-08-22at_13.58.472x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Simply turn on the ones you need.
When subscription-related events happen, Adapty sends events to Airbridge. After receiving them Airbridge sends attribution result information to Adapty. The historical events will be sent in the last 24 hours instead of the real event time

SDK configuration

For the integration, you should pass `airbridge_device_id` to profile builder and call `updateProfile` as it is shown in the example below:

```swift import AirBridge

let builder = AdaptyProfileParameters.Builder() .with(airbridgeDeviceId: AirBridge.deviceUUID())

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin Airbridge.getDeviceInfo().getUUID(object: AirbridgeCallback.SimpleCallback() { override fun onSuccess(result: String) { val params = AdaptyProfileParameters.Builder() .withAirbridgeDeviceId(result) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } } override fun onFailure(throwable: Throwable) { } }) </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:airbridge/*flutter*sdk/airbridge/*flutter*sdk.dart';

final builder = AdaptyProfileParametersBuilder() ..setAirbridgeDeviceId( await Airbridge.state.deviceUUID, );

try { await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import Airbridge from 'airbridge-react-native-sdk'; import { adapty } from 'react-native-adapty';

try { const deviceId = await Airbridge.state.deviceUUID();

await adapty.updateProfile({ airbridgeDeviceId: deviceId, }); } catch (error) { // handle `AdaptyError` } ```

Read more about airbridgeDeviceId in Airbridge documentation.

---

title: "Airbridge" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Airbridge offers an integrated marketing performance analysis for websites and mobile apps by consolidating data collected from multiple devices, platforms, and channels. Using Airbridge's Identity Resolution Engine, you can combine scattered customer identity data from web and app interactions into a unified people-based identity, resulting in more accurate attribution.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

The integration between Adapty and Airbridge operates in two main ways.

1. **Receiving attribution data from Airbridge**
   Once you've set up the Airbridge integration, Adapty will start receiving attribution data from Airbridge. You can easily access and view this data on the user's page.
2. **Sending subscription events to Airbridge**
   Adapty can send all subscription events which are configured in your integration to Airbridge. As a result, you'll be able to track these events within the Airbridge dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

## How to set up Airbridge integration

To integrate Airbridge go to Integrations > Airbridge, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between your Airbridge and Adapty profiles. Airbridge app name and Airbridge API token are required.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

They both can be found in your Airbridge dashboard in the Third-party Integrations > Adapty section.

./2023-02-21at*11.19.29*AM.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Adapty API token field is pre-generated on the Adapty backend. You will need to copy the value of Adapty API token and paste it into the Airbridge Dashboard in the Adapty Authorization Token field.

./2023-03-01at_17.11.412x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

## Events and tags

Below the credentials, there are three groups of events you can send to Airbridge from Adapty

./2023-08-22at_13.58.472x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Simply turn on the ones you need.
When subscription-related events happen, Adapty sends events to Airbridge. After receiving them Airbridge sends attribution result information to Adapty. The historical events will be sent in the last 24 hours instead of the real event time

## SDK configuration

For the integration, you should pass `airbridge_device_id` to profile builder and call `updateProfile` as it is shown in the example below:

```swift import AirBridge

let builder = AdaptyProfileParameters.Builder() .with(airbridgeDeviceId: AirBridge.deviceUUID())

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin Airbridge.getDeviceInfo().getUUID(object: AirbridgeCallback.SimpleCallback() { override fun onSuccess(result: String) { val params = AdaptyProfileParameters.Builder() .withAirbridgeDeviceId(result) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } } override fun onFailure(throwable: Throwable) { } }) </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:airbridge/*flutter*sdk/airbridge/*flutter*sdk.dart';

final builder = AdaptyProfileParametersBuilder() ..setAirbridgeDeviceId( await Airbridge.state.deviceUUID, );

try { await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import Airbridge from 'airbridge-react-native-sdk'; import { adapty } from 'react-native-adapty';

try { const deviceId = await Airbridge.state.deviceUUID();

await adapty.updateProfile({ airbridgeDeviceId: deviceId, }); } catch (error) { // handle `AdaptyError` } ```

Read more about airbridgeDeviceId in Airbridge documentation.

---

title: "Apple Search Ads" description: ""

## metadataTitle: ""

Adapty can help you get attribution data from Apple Search Ads and analyze your metrics with campaign and keyword segmentation. Adapty collects the attribution data for Apple Search Ads automatically through its SDK and AdServices Framework.

Once you've set up the Apple Search Ads integration, Adapty will start receiving attribution data from Apple Search Ads. You can easily access and view this data on the profiles page.

2023-08-21at_15.14.592x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

There are two ways to get attribution: with the old iAd framework and the modern AdServices framework (iOS 14.3+).

## AdServices framework

Apple Search Ads via AdServices does require some configuration in Adapty Dashboard, and you will also need to enable it on the app side. To set up Apple Search Ads using the AdServices framework through Adapty, follow these steps:

### Step 1: Configure Info.plist

Add `AdaptyAppleSearchAdsAttributionCollectionEnabled` to the appâ€™s `Info.plist` file and set it to `YES` (boolean value).

### Step 2: Obtain Public Key

In the Adapty Dashboard, navigate to Settings -> Apple Search Ads.
Locate the pre-generated public key (Adapty provides a key pair for you) and copy it.

2023-08-21at_14.55.542x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note Note: If you're using an alternative service or your own solution for Apple Search Ads attribution, you can upload your own private key. :::

### Step 3: Configure User Management on Apple Search Ads

In your Apple Search Ads account go to Settings > User Management page. In order for Adapty to fetch attribution data you need to invite another Apple ID account and grant it API Account Manager access.

/ width: '700px', / image width / display: 'block', / for alignment / margin: '0 auto' / center alignment */ }} />

#### Step 4: Generate API Credentials

As a next step, log in to the newly added account in Apple Search Ads. Navigate to Settings -> API in the Apple Search Ads interface. Paste the previously copied public key into the designated field. Generate new API credentials.

### Step 5: Configure Adapty with Apple Search Ads Credentials

Copy the Client ID, Team ID, and Key ID fields from the Apple Search Ads settings. In the Adapty Dashboard, paste these credentials into the corresponding fields.

2023-08-21at_15.08.512x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

## iAd Framework

:::danger Deprecated since iOS 14.5

This is the old way to get Search Ads attribution and it is only supported by Adapty SDK prior to version 2.8.0. For more modern approach look into AdServices Framework. :::

Apple Search Ads via iAd Framework doesn't require any configuration in Adapty Dashboard, but you will need to enable it on the app side. Just add `AdaptyAppleSearchAdsAttributionCollectionEnabled` to the appâ€™s `Info.plist` file and set it to `YES` (boolean value).

## Uploading your own keys

:::note Optional

These steps are not required for Apple Search Ads attribution, only for working with other services like Asapty or your own solution. :::

You can use your own public-private key pair if you are using other services or own solution for ASA attribution.

**Step 1**

Generate private key in Terminal

```
text title="Text" openssl ecparam -genkey -name prime256v1 -noout -out private-key.pem
```

Upload it in Adapty Settings -> Apple Search Ads (Upload private key button)

Step 2

Generate public key in Terminal

```
text title="Text" openssl ec -in private-key.pem -pubout -out public-key.pem
```

You can use this public key in your Apple Search Ads settings of account with API Account Manager role. So you can use generated Client ID, Team ID, and Key ID values for Adapty and other services.

---

title: "AppsFlyer" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

AppsFlyer is a leading platform for mobile attribution and marketing analytics. It stands as a third-party service that gathers and organizes data from marketing campaigns. This helps companies see how well their campaigns are performing in one place.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in AppsFlyer and analyze precisely how much revenue your campaigns generate.

The integration between Adapty and AppsFlyer operates in two main ways.

1. **Receiving attribution data from AppsFlyer**
   Once you've set up the AppsFlyer integration, Adapty will start receiving attribution data from AppsFlyer. You can easily access and view this data on the user's profile page.

2023-08-04at_16.29.202x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. **Sending subscription events to AppsFlyer**
   Adapty can send all subscription events that are configured in your integration to AppsFlyer. As a result, you'll be able to track these events within the AppsFlyer dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

How to set up AppsFlyer integration

To setup the integration with AppsFlyer:

1. Open **Integrations -> AppsFlyer** in the Adapty Dashboard.
2. Turn on the toggle to enable the integration.
3. The next step of the integration is to set credentials. To find App ID, open your app page in App Store Connect, go to the **App Information** page in section **General**, and find **Apple ID** in the left bottom part of the screen.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

4. Paste the copied **Apple ID** to the **iOS App ID** in the Adapty Dashboard.

iOSapp_id.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

ðŸ§ If you use AppsFlyer API 2, you need to switch to API 3, since the previous version will be deprecated by AppsFlyer soon. To do so, in the **AppsFlyer S2S API** list, select **API 3**.

5. Open the AppsFlyer site and log in.
6. Click **Your account name** -> **Security Center** in the top-right corner of the dashboard.

securitycenter.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

7. In the **Manage your account security** window, click the **Manage your AppsFlyer API and S2S tokens** button.
8. If you have an S2S token, please proceed to step 12. If you do not have it, click the **New token** button.

newtoken.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

9. In the **New token** window, enter the name of the token. This name is solely for your reference.
10. Choose **S2S** in the **Choose type** list.
11. Click the **Create new token** button to save the new token.
12. In the **Tokens** window, copy the S2S token.
13. In the Adapty Dashboard, paste the copied S2S key into the **Dev key for iOS** and **Dev key for Android** fields.

devkeys.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

14. Click the **Save** button to save the changes. > ðŸ“˜ AppsFlyer doesn't have a Sandbox mode for server2server integration. So you need a different application/account in AppsFlyer for Sandbox Dev Key. If you want to send sandbox events to the same app, just use the same key for production and sandbox.

Adapty maps some events to AppsFlyer standard events by default. With such a configuration, AppsFlyer can then forward events to each ad network that you use without additional setup.

Another important thing is that AppsFlyer doesn't support events older than 24 hours. So, if you have an event that is more than a day old, Adapty will send it to Appsflyer, but the event date and time will be replaced by the current timestamp.

Events and tags

Below the credentials, there are three groups of events you can send to AppsFlyer from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-11at_14.56.362x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to AppsFlyer using a server-to-server integration, allowing you to view all subscription events in your AppsFlyer dashboard and link them to your acquisition campaigns.

SDK configuration

It's very important to send AppsFlyer attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

```
swift // Find your implementation of AppsFlyerLibDelegate // and update onConversionDataSuccess method: func onConversionDataSuccess(_ installData: [AnyHashable : Any]) { // It's important to include the network user ID Adapty.updateAttribution(installData, source: .appsflyer, networkUserId: AppsFlyerLib.shared().getAppsFlyerUID()) } kotlin val conversionListener: AppsFlyerConversionListener = object : AppsFlyerConversionListener { override fun onConversionDataSuccess(conversionData: Map<String, Any>) { // It's important to include the network user ID Adapty.updateAttribution( conversionData, AdaptyAttributionSource.APPSFLYER, AppsFlyerLib.getInstance().getAppsFlyerUID(context) ) { error -> if (error != null) { //handle error } } } } ```javascript import 'package:appsflyer*sdk/appsflyer*sdk.dart';
```

AppsflyerSdk appsflyerSdk = AppsflyerSdk(); appsflyerSdk.onInstallConversionData((data) async { try { // It's important to include the network user ID final appsFlyerUID = await appsFlyerSdk.getAppsFlyerUID(); await Adapty().updateAttribution(data, source: AdaptyAttributionSource.appsflyer, networkUserId: appsFlyerUID,); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} });

appsflyerSdk.initSdk(registerConversionDataCallback: true, registerOnAppOpenAttributionCallback: true, registerOnDeepLinkingCallback: true,); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp using AppsFlyerSDK;

// before SDK initialization AppsFlyer.getConversionData(this.name);

// in your IAppsFlyerConversionData void onConversionDataSuccess(string conversionData) { // It's important to include the network user ID string appsFlyerId = AppsFlyer.getAppsFlyerId(); Adapty.UpdateAttribution(conversionData, AttributionSource.Appsflyer, appsFlyerId, (error) => { // handle the error }); } `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import { adapty, AttributionSource } from 'react-native-adapty'; import appsFlyer from 'react-native-appsflyer';

appsFlyer.onInstallConversionData(installData => { try { // It's important to include the network user ID const networkUserId = appsFlyer.getAppsFlyerUID(); adapty.updateAttribution(installData, AttributionSource.AppsFlyer, networkUserId); } catch (error) { // handle error } });

// ... appsFlyer.initSdk(/.../); ```

---

title: "AppsFlyer" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

AppsFlyer is a leading platform for mobile attribution and marketing analytics. It stands as a third-party service that gathers and organizes data from marketing campaigns. This helps companies see how well their campaigns are performing in one place.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in AppsFlyer and analyze precisely how much revenue your campaigns generate.

The integration between Adapty and AppsFlyer operates in two main ways.

1. **Receiving attribution data from AppsFlyer**
   Once you've set up the AppsFlyer integration, Adapty will start receiving attribution data from AppsFlyer. You can easily access and view this data on the user's profile page.

2023-08-04at_16.29.202x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

2. **Sending subscription events to AppsFlyer**
   Adapty can send all subscription events that are configured in your integration to AppsFlyer. As a result, you'll be able to track these events within the AppsFlyer dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

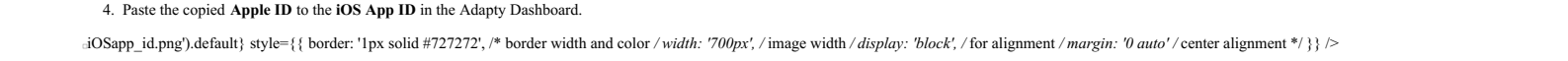
How to set up AppsFlyer integration

To setup the integration with AppsFlyer:

1. Open **Integrations -> AppsFlyer** in the Adapty Dashboard.
2. Turn on the toggle to enable the integration.
3. The next step of the integration is to set credentials. To find App ID, open your app page in App Store Connect, go to the **App Information** page in section **General**, and find **Apple ID** in the left bottom part of the screen.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

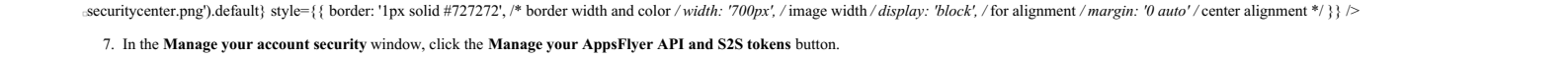
4. Paste the copied **Apple ID** to the **iOS App ID** in the Adapty Dashboard.


iOSapp_id.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

ðŸ§ If you use AppsFlyer API 2, you need to switch to API 3, since the previous version will be deprecated by AppsFlyer soon. To do so, in the **AppsFlyer S2S API** list, select **API 3**.

5. Open the AppsFlyer site and log in.

6. Click **Your account name** -> **Security Center** in the top-right corner of the dashboard.


securitycenter.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

7. In the **Manage your account security** window, click the **Manage your AppsFlyer API and S2S tokens** button.

8. If you have an S2S token, please proceed to step 12. If you do not have it, click the **New token** button.


newtoken.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

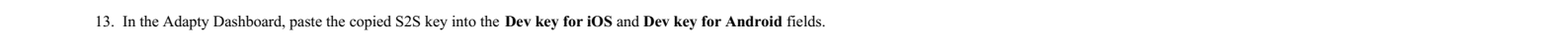
9. In the **New token** window, enter the name of the token. This name is solely for your reference.

10. Choose **S2S** in the **Choose type** list.

11. Click the **Create new token** button to save the new token.

12. In the **Tokens** window, copy the S2S token.

13. In the Adapty Dashboard, paste the copied S2S key into the **Dev key for iOS** and **Dev key for Android** fields.


devkeys.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

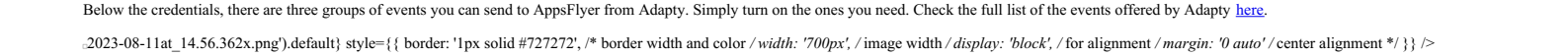
14. Click the **Save** button to save the changes. > ðŸ“˜ AppsFlyer doesn't have a Sandbox mode for server2server integration. So you need a different application/account in AppsFlyer for Sandbox Dev Key. If you want to send sandbox events to the same app, just use the same key for production and sandbox.

Adapty maps some events to AppsFlyer standard events by default. With such a configuration, AppsFlyer can then forward events to each ad network that you use without additional setup.

Another important thing is that AppsFlyer doesn't support events older than 24 hours. So, if you have an event that is more than a day old, Adapty will send it to Appsflyer, but the event date and time will be replaced by the current timestamp.

Events and tags

Below the credentials, there are three groups of events you can send to AppsFlyer from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.


2023-08-11at_14.56.362x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to AppsFlyer using a server-to-server integration, allowing you to view all subscription events in your AppsFlyer dashboard and link them to your acquisition campaigns.

SDK configuration

It's very important to send AppsFlyer attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

```
swift // Find your implementation of AppsFlyerLibDelegate // and update onConversionDataSuccess method: func onConversionDataSuccess(_ installData: [AnyHashable : Any]) { // It's important to include the network user ID Adapty.updateAttribution(installData, source: .appsflyer, networkUserId: AppsFlyerLib.shared().getAppsFlyerUID()) } kotlin val conversionListener: AppsFlyerConversionListener = object : AppsFlyerConversionListener { override fun onConversionDataSuccess(conversionData: Map<String, Any>) { // It's important to include the network user ID Adapty.updateAttribution( conversionData, AdaptyAttributionSource.APPSFLYER, AppsFlyerLib.getInstance().getAppsFlyerUID(context) ) { error -> if (error != null) { //handle error } } } } ```javascript import 'package:appsflyer_sdk/appsflyer_sdk.dart';

AppsflyerSdk appsflyerSdk = AppsflyerSdk(); appsflyerSdk.onInstallConversionData((data) async { try { // It's important to include the network user ID final appsFlyerUID = await appsFlyerSdk.getAppsFlyerUID(); await Adapty().updateAttribution( data, source: AdaptyAttributionSource.appsflyer, networkUserId: appsFlyerUID, ); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} });

appsflyerSdk.initSdk( registerConversionDataCallback: true, registerOnAppOpenAttributionCallback: true, registerOnDeepLinkingCallback: true, ); </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp using AppsFlyerSDK;

// before SDK initialization AppsFlyer.getConversionData(this.name);

// in your IAppsFlyerConversionData void onConversionDataSuccess(string conversionData) { // It's important to include the network user ID string appsFlyerId = AppsFlyer.getAppsFlyerId(); Adapty.UpdateAttribution(conversionData, AttributionSource.Appsflyer, appsFlyerId, (error) => { // handle the error }); } </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty, AttributionSource } from 'react-native-adapty'; import appsFlyer from 'react-native-appsflyer';

appsFlyer.onInstallConversionData(installData => { try { // It's important to include the network user ID const networkUserId = appsFlyer.getAppsFlyerUID(); adapty.updateAttribution(installData, AttributionSource.AppsFlyer, networkUserId); } catch (error) { // handle error } });

// ... appsFlyer.initSdk(/.../); ```

---

title: "Switch from AppsFlyer S2S API 2 to 3" description: "Learn how Adapty's support for AppsFlyer S2S API 3 provides a seamless transition from API 2, enhancing security and reducing fraud in in-app events. Switch today"

## metadataTitle: "Adapty Supports AppsFlyer S2S API 3 Upgrade: Enhanced Security and Fraud Reduction"

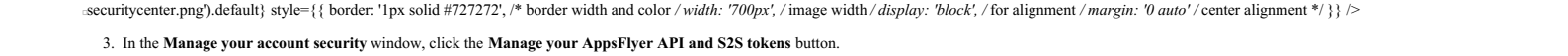
According to the official AppsFlyer What's New, to provide a more secure experience for API usage and to reduce fraud, AppsFlyer has upgraded its server-to-server (S2S) API for in-app events. The existing endpoint will be deprecated in the future and we recommend to start planning the switch.

Adapty supports AppsFlyer S2S API 3 and provides you with a seamless switch from API 2. Keep in mind that this switch is one-way, so you wonâ€™t be able to return to API 2 once youâ€™ve made the change.

To switch from AppsFlyer S2S API 2 to 3:

1. Open the AppsFlyer site and log in.

2. Click **Your account name** -> **Security Center** in the top-left corner of the dashboard.


securitycenter.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

3. In the **Manage your account security** window, click the **Manage your AppsFlyer API and S2S tokens** button.

4. If you do not have an S2S token, click the **New token** button. If you have it, please proceed with step 8.


newtoken.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

5. In the **New token** window, enter the name of the token. This name is solely for your reference.

6. Choose **S2S** in the **Choose type** list.

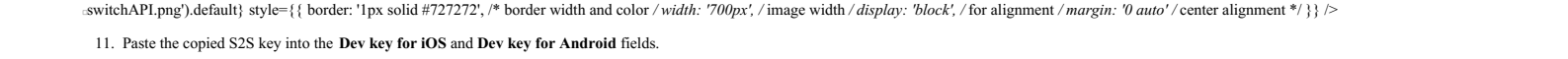
7. Don't forget to click the **Create new token** button to save the new token.

8. In the **Tokens** window, copy the S2S token.

./ width: '700px', / image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

9. Open **Integrations -> AppsFlyer** in the Adapty Dashboard.

10. In the **AppsFlyer S2S API** field, select **API 3**.


switchAPI.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px', /* image width / *display: 'block', /* for alignment / *margin: '0 auto'* / center alignment */ }} />

11. Paste the copied S2S key into the **Dev key for iOS** and **Dev key for Android** fields.
```

12. Click the **Save** button to confirm the switch.

At this moment, your integration instantly switches to AppsFlyer S2S API 3 and your new events will be sent to the new URL.

---

title: "Asapty" description: ""

metadataTitle: ""

Using Asapty integration you can optimize your Search Ads campaigns. Adapty sends subscription events to Asapty, so you can build custom dashboards there, based on Apple Search Ads attribution.

This specific integration doesn't add any attribution data to Adapty, as we already have everything we need from ASA directly.

How to set up Asapty integration

To integrate Asapty navigate to Integrations > Asapty in the Adapty dashboard and fill out the field value for Asapty ID.

2023-08-14at_18.57.462x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Asapty ID can be found in Settings> General section in your Asapty account.

Events and tags

Below the credentials, there are three groups of events you can send to Asapty from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-15at_15.11.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Asapty. But you can change the event names based on your needs.

SDK configuration

You don't have to configure anything on the SDK side, but we recommend sending `customerUserId` to Adapty for better accuracy.

:::warning Troubleshooting

- Make sure you've configured Apple Search Ads in Adapty and uploaded credentials, without them, Asapty won't work.
- Only the profiles with detailed, non-organic ASA attribution will deliver their events to Asapty. You will see "The user profile is missing the required integration data." if the attribution is no sufficients.
- Profiles created prior to configuring the integrations will not be able to deliver their events to Asapty. You will see "The user profile is missing the required integration data." error in such cases. :::

---

title: "Asapty" description: ""

metadataTitle: ""

Using Asapty integration you can optimize your Search Ads campaigns. Adapty sends subscription events to Asapty, so you can build custom dashboards there, based on Apple Search Ads attribution.

This specific integration doesn't add any attribution data to Adapty, as we already have everything we need from ASA directly.

How to set up Asapty integration

To integrate Asapty navigate to Integrations > Asapty in the Adapty dashboard and fill out the field value for Asapty ID.

2023-08-14at_18.57.462x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Asapty ID can be found in Settings> General section in your Asapty account.

Events and tags

Below the credentials, there are three groups of events you can send to Asapty from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-15at_15.11.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Asapty. But you can change the event names based on your needs.

SDK configuration

You don't have to configure anything on the SDK side, but we recommend sending `customerUserId` to Adapty for better accuracy.

:::warning Troubleshooting

- Make sure you've configured Apple Search Ads in Adapty and uploaded credentials, without them, Asapty won't work.
- Only the profiles with detailed, non-organic ASA attribution will deliver their events to Asapty. You will see "The user profile is missing the required integration data." if the attribution is no sufficients.
- Profiles created prior to configuring the integrations will not be able to deliver their events to Asapty. You will see "The user profile is missing the required integration data." error in such cases. :::

---

title: "Branch" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Branch enables customers to reach, interact, and assess results across diverse devices, channels, and platforms. It's a user-friendly platform designed to enhance mobile revenue through specialized links that work seamlessly on all devices, channels, and platforms.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

The integration between Adapty and Branch operates in two main ways.

1. **Receiving attribution data from Branch**
   Once you've set up the Branch integration, Adapty will start receiving attribution data from Branch. You can easily access and view this data on the user's profile page.

2023-08-11at_17.36.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. **Sending subscription events to Branch**
   Adapty can send all subscription events which are configured in your integration to Branch. As a result, you'll be able to track these events within the Branch dashboard.

How to set up Branch integration

To integrate Branch go to Integrations > Branch in Adapty Dashboard , turn on a toggle from off to on, and fill out fields.

2023-08-11at_15.54.372x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

To get the value for the **Branch Key**, open your Branch Account Settings and find the **Branch Key** field. Use it for the **Key test** or **Key live** field in the Adapty Dashboard. In Branch, switch between Live and Tests environments for the appropriate key.

2023-08-11at_15.24.162x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Events and tags

Below the credentials, there are three groups of events you can send to Branch from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

You can send an event with Proceeds (after Apple/Google cut) or just revenue. Also, you can check a box for reporting in the user's currency.

2023-08-11at_15.18.282x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to Branch using a server-to-server integration, allowing you to view all subscription events in your Branch dashboard and link them to your acquisition campaigns.

SDK configuration

It's very important to send Branch attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

To connect the Branch and Adapty user, make sure you provide your `customerUserId` as Branch Identity id. If you prefer not to use `customerUserId` in Branch, use `networkUserId` param in attribution method to specify the Branch user ID to attach to.

`swift // login Branch.getInstance().setIdentity("YOUR_USER_ID")` ```kotlin // login and update attribution Branch.getAutoInstance(this) .setIdentity("YOUR*USER*ID") { referringParams, error -> referringParams?.let { params -> Adapty.updateAttribution(data, AdaptyAttributionSource.BRANCH) { error -> if (error != null) { //handle error } } } }

// logout Branch.getAutoInstance(context).logout() `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript import 'package:flutter*branch*sdk/flutter*branch*sdk.dart';

FlutterBranchSdk.setIdentity('YOUR*USER*ID'); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp Branch.setIdentity("your user id"); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import branch from 'react-native-branch';

branch.setIdentity('YOUR*USER*ID'); ```

Next, pass the attribution you receive from the initializing method of Branch iOS SDK to Adapty.

`swift // Pass the attribution you receive from the initializing method of Branch iOS SDK to Adapty. Branch.getInstance().initSession(launchOptions: launchOptions) { (data, error) in if let data = data { Adapty.updateAttribution(data, source: .branch) } } kotlin //everything is in the above snippet for Android` ```javascript import 'package:flutter*branch*sdk/flutter*branch*sdk.dart';

FlutterBranchSdk.initSession().listen((data) async { try { await Adapty().updateAttribution(data, source: AdaptyAttributionSource.branch); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} }); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp Branch.initSession(delegate(Dictionary parameters, string error) { string attributionString = JsonUtility.ToJson(parameters); Adapty.UpdateAttribution(attributionString, AttributionSource.Branch, (error) => { // handle the error }); }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import { adapty, AttributionSource } from 'react-native-adapty'; import branch from 'react-native-branch';

branch.subscribe({ enComplete: ({ params, }) => { adapty.updateAttribution(params, AttributionSource.Branch); }, }); ```

---

title: "Branch" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Branch enables customers to reach, interact, and assess results across diverse devices, channels, and platforms. It's a user-friendly platform designed to enhance mobile revenue through specialized links that work seamlessly on all devices, channels, and platforms.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

The integration between Adapty and Branch operates in two main ways.

1. **Receiving attribution data from Branch**
   Once you've set up the Branch integration, Adapty will start receiving attribution data from Branch. You can easily access and view this data on the user's profile page.

2023-08-11at_17.36.072x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. **Sending subscription events to Branch**
   Adapty can send all subscription events which are configured in your integration to Branch. As a result, you'll be able to track these events within the Branch dashboard.

How to set up Branch integration

To integrate Branch go to Integrations > Branch in Adapty Dashboard , turn on a toggle from off to on, and fill out fields.

2023-08-11at_15.54.372x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

To get the value for the **Branch Key**, open your Branch Account Settings and find the **Branch Key** field. Use it for the **Key test** or **Key live** field in the Adapty Dashboard. In Branch, switch between Live and Tests environments for the appropriate key.

2023-08-11at_15.24.162x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Events and tags

Below the credentials, there are three groups of events you can send to Branch from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

You can send an event with Proceeds (after Apple/Google cut) or just revenue. Also, you can check a box for reporting in the user's currency.

2023-08-11at_15.18.282x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to Branch using a server-to-server integration, allowing you to view all subscription events in your Branch dashboard and link them to your acquisition campaigns.

SDK configuration

It's very important to send Branch attribution data from the device to Adapty using `Adapty.updateAttribution()` SDK method. The example below shows how to do that.

To connect the Branch and Adapty user, make sure you provide your `customerUserId` as Branch Identity id. If you prefer not to use `customerUserId` in Branch, use `networkUserId` param in attribution method to specify the Branch user ID to attach to.

`swift // login Branch.getInstance().setIdentity("YOUR_USER_ID")` ```kotlin // login and update attribution Branch.getAutoInstance(this) .setIdentity("YOUR*USER*ID") { referringParams, error -> referringParams?.let { params -> Adapty.updateAttribution(data, AdaptyAttributionSource.BRANCH) { error -> if (error != null) { //handle error } } } }

// logout Branch.getAutoInstance(context).logout() `</TabItem> <TabItem value="Flutter" label="Flutter" default>` javascript import 'package:flutter*branch*sdk/flutter*branch*sdk.dart';

FlutterBranchSdk.setIdentity('YOUR*USER*ID'); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp Branch.setIdentity("your user id"); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import branch from 'react-native-branch';

branch.setIdentity('YOUR*USER*ID'); ```

Next, pass the attribution you receive from the initializing method of Branch iOS SDK to Adapty.

`swift // Pass the attribution you receive from the initializing method of Branch iOS SDK to Adapty. Branch.getInstance().initSession(launchOptions: launchOptions) { (data, error) in if let data = data { Adapty.updateAttribution(data, source: .branch) } } kotlin //everything is in the above snippet for Android` ```javascript import 'package:flutter*branch*sdk/flutter*branch*sdk.dart';

FlutterBranchSdk.initSession().listen((data) async { try { await Adapty().updateAttribution(data, source: AdaptyAttributionSource.branch); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} }); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp Branch.initSession(delegate(Dictionary parameters, string error) { string attributionString = JsonUtility.ToJson(parameters); Adapty.UpdateAttribution(attributionString, AttributionSource.Branch, (error) => { // handle the error }); }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import { adapty, AttributionSource } from 'react-native-adapty'; import branch from 'react-native-branch';

branch.subscribe({ enComplete: ({ params, }) => { adapty.updateAttribution(params, AttributionSource.Branch); }, }); ```

---

title: "Facebook Ads" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

With the Facebook Ads integration, you can easily check your app stats on Facebook Analytics. Adapty sends events to Facebook Ads Manager, helping you make similar audiences based on subscriptions to get better returns. This way, you can accurately see how much money your ads are making from subscriptions.

The integration between Adapty and Facebook Ads operates in the following way: Adapty sends all subscription events that are configured in your integration to Facebook Ads. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

How to set up Facebook Ads integration

To integrate Facebook Ads and analyze your app metrics, you can set up the integration with Facebook Analytics. By sending events to Facebook Ads Manager, you can create lookalike audiences based on subscription events like renewals. To configure this integration, navigate to Integrations > Facebook Ads in the Adapty Dashboard and provide the required credentials.

:::note Please consider that Facebook Ads integration works on iOS 14.5+ only for users with ATT consent. :::

2023-08-15at_15.45.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

1. To find App ID, open your app page in App Store Connect, go to the **App Information** page in section **General**, and find **Apple ID** in the left bottom part of the screen.
2. You need an application on Facebook Developers platform. Log in to your app and then find advanced settings. You can find the **App ID** in the header.

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

:::warning Disable in-app events logging in the Facebook SDK to avoid duplications

Open your App Dashboard and navigate to Analytics->Settings. Then set *Log In-App Events Automatically* to *No* and click *Save Changes*. :::

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

You can use this integration with Android apps as well. If you set up Android SDK configuration in the App Settings, setting up the Facebook App ID is enough.

Events and tags

Please note that the Facebook Ads integration specifically caters to companies using Facebook for ad campaigns and optimizing them based on customer behavior. It supports Facebook's standard events for optimization purposes. Consequently, modifying the event name is not available for the Facebook Ads integration. Adapty effectively maps your customer events to their corresponding Facebook events for accurate analysis.

| Adapty event | Facebook Ads event | | :---------- | :---------- | | Subscription initial purchase | Subscribe | | Subscription renewed | Subscribe | | Subscription cancelled | CancelSubscription | | Trial started | StartTrial | | Trial converted | Subscribe | | Trial cancelled | CancelTrial | | Non subscription purchase | fb*mobile*purchase | | Billing issue detected | billing*issue*detected | | Entered grace period | entered*grace*period | | Auto renew off | auto*renew*off | | Auto renew on | auto*renew*on | | Auto renew off subscription | auto*renew*off*subscription* | | *Auto renew on subscription* | *auto*renew*on*subscription |

StartTrial, Subscribe, CancelSubscription are standard events.

2023-07-04at_12.47.312x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

To enable specific events, simply toggle on the ones you require. In case multiple event names are selected, Adapty will consolidate the data from all the chosen events into a single Adapty event name.

SDK configuration

:::warning Because of iOS IDFA changes in iOS 14.5, if you use Facebook integration, make sure you send `facebookAnonymousId` to Adapty via `.updateProfile()` method. It allows Facebook to handle events if IDFA is not available. :::

```swift import FacebookCore

let builder = AdaptyProfileParameters.Builder() .with(facebookAnonymousId: AppEvents.shared.anonymousID)

Adapty.updateProfile(params: builder.build()) { error in if error != nil { // handle the error
} } </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val builder = AdaptyProfileParameters.Builder()
.withFacebookAnonymousId(AppEventsLogger.getAnonymousAppDeviceGUID(context))

Adapty.updateProfile(builder.build()) { error -> if (error == null) { // successful update } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> text There is no official SDK for Flutter </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp anonymousID is not available in the official SDK https://github.com/facebook/facebook-sdk-for-unity/issues/676 </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import { AppEventsLogger } from 'react-native-fbsdk-next';

try { const anonymousId = await AppEventsLogger.getAnonymousID();

await adapty.updateProfile({ facebookAnonymousId: anonymousId, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "Singular" description: "Learn how to set up integration with Singular"

## metadataTitle: "Singular"

Singular is one of the leading Mobile Measurement Partner (MMP) platforms, that collects and presents data from marketing campaigns. This helps companies track their campaign performance.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in Singular and analyze precisely how much revenue your campaigns generate.

Adapty can send all subscription events which are configured in your integration to Singular. As a result, you'll be able to track these events within the Singular dashboard. This integration is beneficial for evaluating the effectiveness of your advertising campaigns.

## How to set up Singular integration

To set up the integration with AppsFlyer, go to Integrations > Singular in your Adapty Dashboard, turn on a toggle, and fill out the fields.

For this integration to work, the Singular SDK Key is required. It can be found in Singular dashboard: Developer tools -> SDK Keys -> SDK Key (**not** SDK Secret):

sdkkey.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Below the credentials, there are three groups of events you can send to Singular from Adapty. Check the full list of the events offered by Adapty here.

/ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to Singular using a server-to-server integration, allowing you to view all subscription events in your Singular dashboard and link them to your acquisition campaigns.

:::warning Profiles created prior to configuring the integrations will not be able to deliver their events to Singular. You will see "The user profile is missing the required integration data." error in such cases. :::

## No need for SDK configuration

There is no need to configure the SDK from your side at the moment â€” as Adapty already collects the data required by Singular and this integration is server-to-server. In case it ever changes, we'll let you know.

---

title: "Singular" description: "Learn how to set up integration with Singular"

## metadataTitle: "Singular"

Singular is one of the leading Mobile Measurement Partner (MMP) platforms, that collects and presents data from marketing campaigns. This helps companies track their campaign performance.

Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in Singular and analyze precisely how much revenue your campaigns generate.

Adapty can send all subscription events which are configured in your integration to Singular. As a result, you'll be able to track these events within the Singular dashboard. This integration is beneficial for evaluating the
```

effectiveness of your advertising campaigns.

How to set up Singular integration

To set up the integration with AppsFlyer, go to Integrations > Singular in your Adapty Dashboard, turn on a toggle, and fill out the fields.

For this integration to work, the Singular SDK Key is required. It can be found in Singular dashboard: Developer tools -> SDK Keys -> SDK Key (**not** SDK Secret):

sdkkey.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Below the credentials, there are three groups of events you can send to Singular from Adapty. Check the full list of the events offered by Adapty here.

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

Adapty will send subscription events to Singular using a server-to-server integration, allowing you to view all subscription events in your Singular dashboard and link them to your acquisition campaigns.

:::warning Profiles created prior to configuring the integrations will not be able to deliver their events to Singular. You will see "The user profile is missing the required integration data." error in such cases. :::

No need for SDK configuration

There is no need to configure the SDK from your side at the moment â€” as Adapty already collects the data required by Singular and this integration is server-to-server. In case it ever changes, we'll let you know.

---

title: "Analytics integrations" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty sends all subscription events to analytical services, such as Amplitude, Mixpanel, and AppMetrica. We can also send events to your server using webhook integration. The best thing about this is that you don't have to send any of the events, we'll do it for you. Just make sure to configure the integration in the Adapty Dashboard.

Adapty supports the integration with the following 3d-party analytics services:

- Amplitude
- AppMetrica
- Firebase and Google Analytics
- Mixpanel
- Slack

:::note Don't see your analytics provider?

Let us know! Write to the Adapty support and we'll consider adding it. :::

Setting the profile's identifier

Set the profile's identifier for the selected analytics using `.updateProfile()` method. For example, for Amplitude integration, you can set either `amplitudeUserId` or `amplitudeDeviceId`. For Mixpanel integration, you have to set `mixpanelUserId`. When these identifiers are not set, Adapty will use `customerUserId` instead. If the `customerUserId` is not set, we will use our internal profile id.

:::warning Avoiding duplication

Don't forget to turn off sending subscription events from devices and your server to avoid duplication :::

Disabling external analytics for a specific customer

You may want to stop sending analytics events for a specific customer. This is useful if you have an option in your app to opt-out of analytics services.

To disable external analytics for a customer, use `updateProfile()` method. Create `AdaptyProfileParameters.Builder` object and set corresponding value to it.
When external analytics is blocked, Adapty won't be sending any events to any integrations for the specific user. If you want to disable an integration for all users of your app, just turn it off in Adapty Dashboard.

```swift let builder = AdaptyProfileParameters.Builder() .with(analyticsDisabled: true)

Adapty.updateProfile(parameters: builder.build()) </TabItem> <TabItem value="kotlin" label="Kotlin" default> kotlin val builder = AdaptyProfileParameters.Builder() .withExternalAnalyticsDisabled(true)

Adapty.updateProfile(builder.build()) </TabItem> <TabItem value="java" label="Java" default> java ] AdaptyProfileParameters.Builder builder = new AdaptyProfileParameters.Builder() .withExternalAnalyticsDisabled(true);

Adapty.updateProfile(builder.build()); </TabItem> <TabItem value="Flutter" label="Flutter" default> javascript final builder = AdaptyProfileParametersBuilder() ..setAnalyticsDisabled(true);

try { await Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { } </TabItem> <TabItem value="Unity" label="Unity" default> csharp var builder = new Adapty.ProfileParameters.Builder() .SetAnalyticsDisabled(true);

Adapty.UpdateProfile(builder.Build(), (error) => { if(error != null) { // handle the error } }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript adapty.updateProfile({ analyticsDisabled: true }); ```

### Disable collection of IDFA

You can disable IDFA collecting by using property `idfaCollectionDisabled`. Make sure you call it before `.activate()` method.

`swift title="Swift" Adapty.idfaCollectionDisabled = true Adapty.activate("YOUR_ADAPTY_APP_TOKEN")` You can disable IDFA collecting by adding specific key to the Adapty-Info.plist file:

`xml title="Adapty-Info.plist" <key>AdaptyIDFACollectionDisabled</key> <true/>` You also can disable IDFA collecting by setting `idfaCollectionDisabled` flag in your activation flow:

`typescript title="Typescript" adapty.activate('PUBLIC_SDK_KEY', { ios: { idfaCollectionDisabled: false, }, });`

---

title: "Amplitude" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Amplitude is a powerful mobile analytics service. With Adapty, you can easily send events to Amplitude, see how users behave, and then make smart decisions.

Adapty provides a complete set of data that lets you track subscription events from stores in one place and sends it to your Amplitude account. This allows you to match your user behavior with their payment history in Amplitude, and inform your product decisions.

### How to set up Amplitude integration

To set up the integration with Amplitude, go to Integrations > Amplitude in the Adapty Dashboard, turn on a toggle from off to on, and fill out fields.


2023-08-15at_16.47.102x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

You need to enter the **API Key** into Adapty. To find a token, go to your **Project settings** in Amplitude. In case you need help, refer to official docs.


2023-08-15at_16.53.512x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Along with events, Adapty also sends the subscription status and subscription product ID to the Amplitude user properties.

### Events and tags

Below the credentials, there are three groups of events you can send to Amplitude from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.


2023-08-15at_16.52.352x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />
```

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs. Adapty will send subscription events to Amplitude using a server-to-server integration, allowing you to view all subscription events in your Amplitude dashboard.

SDK configuration

Use `Adapty.updateProfile()` method to set `amplitudeDeviceId` or `amplitudeUserId`. If not set, Adapty uses your user ID (`customerUserId`) or if it's null Adapty ID. Make sure that the user id you use to send data to Amplitude from your app is the same one you send to Adapty.

```Text import Amplitude

let builder = AdaptyProfileParameters.Builder() .with(amplitudeUserId: Amplitude.instance().userId) .with(amplitudeDeviceId: Amplitude.instance().deviceId)

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin //for Amplitude maintenance SDK (obsolete) val amplitude = Amplitude.getInstance() val amplitudeDeviceId = amplitude.deviceId val amplitudeUserId = amplitude.userId

//for actual Amplitude Kotlin SDK val amplitude = Amplitude( Configuration( apiKey = AMPLITUDE*API*KEY, context = applicationContext ) ) val amplitudeDeviceId = amplitude.store.deviceId val amplitudeUserId = amplitude.store.userId

//

val params = AdaptyProfileParameters.Builder() .withAmplitudeDeviceId(amplitudeDeviceId) .withAmplitudeUserId(amplitudeUserId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:amplitude_flutter/amplitude.dart';

final Amplitude amplitude = Amplitude.getInstance(instanceName: "YOUR*INSTANCE*NAME");

final builder = AdaptyProfileParametersBuilder() ..setAmplitudeDeviceId(await amplitude.getDeviceId()) ..setAmplitudeUserId(await amplitude.getUserId());

try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetAmplitudeUserId("AMPLITUDE*USER*ID"); builder.SetAmplitudeDeviceId(amplitude.getDeviceId());

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty';

try { await adapty.updateProfile({ amplitudeDeviceId: deviceId, amplitudeUserId: userId, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "Amplitude" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Amplitude is a powerful mobile analytics service. With Adapty, you can easily send events to Amplitude, see how users behave, and then make smart decisions.

Adapty provides a complete set of data that lets you track subscription events from stores in one place and sends it to your Amplitude account. This allows you to match your user behavior with their payment history in Amplitude, and inform your product decisions.

### How to set up Amplitude integration

To set up the integration with Amplitude, go to Integrations > Amplitude in the Adapty Dashboard, turn on a toggle from off to on, and fill out fields.

2023-08-15at_16.47.102x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

You need to enter the **API Key** into Adapty. To find a token, go to your **Project settings** in Amplitude. In case you need help, refer to official docs.

2023-08-15at_16.53.512x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

Along with events, Adapty also sends the subscription status and subscription product ID to the Amplitude user properties.

### Events and tags

Below the credentials, there are three groups of events you can send to Amplitude from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-15at_16.52.352x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs. Adapty will send subscription events to Amplitude using a server-to-server integration, allowing you to view all subscription events in your Amplitude dashboard.

### SDK configuration

Use `Adapty.updateProfile()` method to set `amplitudeDeviceId` or `amplitudeUserId`. If not set, Adapty uses your user ID (`customerUserId`) or if it's null Adapty ID. Make sure that the user id you use to send data to Amplitude from your app is the same one you send to Adapty.

```Text import Amplitude

let builder = AdaptyProfileParameters.Builder() .with(amplitudeUserId: Amplitude.instance().userId) .with(amplitudeDeviceId: Amplitude.instance().deviceId)

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin //for Amplitude maintenance SDK (obsolete) val amplitude = Amplitude.getInstance() val amplitudeDeviceId = amplitude.deviceId val amplitudeUserId = amplitude.userId

//for actual Amplitude Kotlin SDK val amplitude = Amplitude( Configuration( apiKey = AMPLITUDE*API*KEY, context = applicationContext ) ) val amplitudeDeviceId = amplitude.store.deviceId val amplitudeUserId = amplitude.store.userId

//

val params = AdaptyProfileParameters.Builder() .withAmplitudeDeviceId(amplitudeDeviceId) .withAmplitudeUserId(amplitudeUserId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:amplitude_flutter/amplitude.dart';

final Amplitude amplitude = Amplitude.getInstance(instanceName: "YOUR*INSTANCE*NAME");

final builder = AdaptyProfileParametersBuilder() ..setAmplitudeDeviceId(await amplitude.getDeviceId()) ..setAmplitudeUserId(await amplitude.getUserId());

try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetAmplitudeUserId("AMPLITUDE*USER*ID"); builder.SetAmplitudeDeviceId(amplitude.getDeviceId());

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty';

try { await adapty.updateProfile({ amplitudeDeviceId: deviceId, amplitudeUserId: userId, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "AppMetrica" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

AppMetrica is a no-cost tool that helps you track advertisements and analyze how your mobile app is doing. It works in real-time, so you see things right away.

### How to set up AppMetrica integration

To integrate AppMetrica go to Integrations > AppMetrica and set credentials.

2023-08-18at_14.57.352x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

Open AppMetrica apps list. Choose the app you want to send events to and go to **Settings**. Copy **Application ID** and **Post API key** and use them to set up the integration in Adapty.

2023-08-18at_19.56.422x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

AppMetrica syncs events every 4 hours, so it may take some time for events to appear in the dashboard. AppMetrica doesn't support sending events revenue, but we send it as regular property.

## Events and tags

Below the credentials, there are three groups of events you can send to AppMetrics from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-18at_14.59.042x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

## SDK configuration

Use `Adapty.updateProfile()` method to set `appmetricaProfileId` or `appmetricaDeviceId`. If not set, Adapty uses your user ID (`customerUserId`). Make sure that the user id you use to send data to AppMetrica from your app is the same one you send to Adapty. These links should help to set up a user id for AppMetrica in your app.

- Set profile ID iOS;
- Get device ID iOS;
- Set profile id Android;
- Get device ID Android.

```swift import YandexMobileMetrica

YMMYandexMetrica.requestAppMetricaDeviceID(withCompletionQueue: .main) { deviceId, error in guard let deviceId = deviceId else { return }

let builder = AdaptyProfileParameters.Builder()
    .with(appmetricaDeviceId: deviceId)
    .with(appmetricaProfileId: "YOUR_ADAPTY_CUSTOMER_USER_ID")

    Adapty.updateProfile(params: builder.build())

} </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withAppmetricaDeviceId(appmetricaDeviceId) .withAppmetricaProfileId(appmetricaProfileId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:appmetrica*plugin/appmetrica*plugin.dart';

final builder = AdaptyProfileParametersBuilder() ..setAppmetricaDeviceId(await AppMetrica.requestAppMetricaDeviceID()) ..setAppmetricaProfileId("YOUR*ADAPTY*CUSTOMER*USER*ID")

try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp AppMetrica.Instance.RequestAppMetricaDeviceID((deviceId, error) => { if (error != null) { // handle error return; }

var builder = new Adapty.ProfileParameters.Builder();

builder.SetAppmetricaProfileId("YOUR_ADAPTY_CUSTOMER_USER_ID");
builder.SetAppmetricaDeviceId(deviceId);

Adapty.UpdateProfile(builder.Build(), (error) => {
    // handle error
});

}); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty';

// ... try { await adapty.updateProfile({ appmetricaProfileId: appmetricaProfileId, appmetricaDeviceId: appmetricaDeviceId, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "AppMetrica" description: ""

# metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

AppMetrica is a no-cost tool that helps you track advertisements and analyze how your mobile app is doing. It works in real-time, so you see things right away.

### How to set up AppMetrica integration

To integrate AppMetrica go to Integrations > AppMetrica and set credentials.

2023-08-18at_14.57.352x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Open AppMetrica apps list. Choose the app you want to send events to and go to **Settings**. Copy **Application ID** and **Post API key** and use them to set up the integration in Adapty.

2023-08-18at_19.56.422x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

AppMetrica syncs events every 4 hours, so it may take some time for events to appear in the dashboard. AppMetrica doesn't support sending events revenue, but we send it as regular property.

## Events and tags

Below the credentials, there are three groups of events you can send to AppMetrics from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-18at_14.59.042x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

## SDK configuration

Use `Adapty.updateProfile()` method to set `appmetricaProfileId` or `appmetricaDeviceId`. If not set, Adapty uses your user ID (`customerUserId`). Make sure that the user id you use to send data to AppMetrica from your app is the same one you send to Adapty. These links should help to set up a user id for AppMetrica in your app.

- Set profile ID iOS;
- Get device ID iOS;
- Set profile id Android;
- Get device ID Android.

```swift import YandexMobileMetrica

YMMYandexMetrica.requestAppMetricaDeviceID(withCompletionQueue: .main) { deviceId, error in guard let deviceId = deviceId else { return }

let builder = AdaptyProfileParameters.Builder()
    .with(appmetricaDeviceId: deviceId)
    .with(appmetricaProfileId: "YOUR_ADAPTY_CUSTOMER_USER_ID")

    Adapty.updateProfile(params: builder.build())

} </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withAppmetricaDeviceId(appmetricaDeviceId) .withAppmetricaProfileId(appmetricaProfileId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:appmetrica*plugin/appmetrica*plugin.dart';

final builder = AdaptyProfileParametersBuilder() ..setAppmetricaDeviceId(await AppMetrica.requestAppMetricaDeviceID()) ..setAppmetricaProfileId("YOUR*ADAPTY*CUSTOMER*USER*ID")

try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp AppMetrica.Instance.RequestAppMetricaDeviceID((deviceId, error) => { if (error != null) { // handle error return; }

var builder = new Adapty.ProfileParameters.Builder();

builder.SetAppmetricaProfileId("YOUR_ADAPTY_CUSTOMER_USER_ID");
builder.SetAppmetricaDeviceId(deviceId);

Adapty.UpdateProfile(builder.Build(), (error) => {
    // handle error
});

}); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty';

// ... try { await adapty.updateProfile({ appmetricaProfileId: appmetricaProfileId, appmetricaDeviceId: appmetricaDeviceId, }); } catch (error) { // handle `AdaptyError` } ```

title: "Firebase and Google Analytics" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

If you use such Google products as Google Analytics, Firebase, and BigQuery you may enrich your analytical data with events from Adapty using the integration described in this article. Events are sent through Google Analytics to Firebase and may be used in any of these services.

## How to set up Firebase integration

### 1. Set up Firebase

First of all, you have to enable integration between Firebase and Google Analytics. You can do it in your Firebase Console in the**Integrations** tab.

2023-08-18at_20.37.462x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

### 2. Integrate with Adapty

Then Adapty needs your Firebase App ID and Google Analytics API Secret to send events and user properties. You can find these parameters in the Firebase Console and Google Analytics Data Streams Tab respectively.

2023-08-21at_12.14.182x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Next, access the App's Stream details page within the Data Streams section of Admin settings in Google Analytics.

2023-08-21at_12.28.482x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Under **Additional settings**, go to the **Measurement Protocol API secrets** page and create a new **API Secret** if it doesn't exist. Copy the value.

2023-08-21at_12.33.242x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2023-08-21at_12.34.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Then, your next step will be adjusting integration in Adapty Dashboard. You will need to provide Firebase App ID and Google Analytics API Secret to us for your iOS and Android platforms.

2023-08-21at_12.35.312x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

## SDK configuration

Then you have to set up Adapty SDK to associate your users with Firebase. For each user, you should send the`firebase_app_instance_id` to Adapty. Here you can see an example of the code which can be used to integrate Firebase SDK and Adapty SDK.

```swift import FirebaseCore import FirebaseAnalytics

FirebaseApp.configure()

if let appInstanceId = Analytics.appInstanceID() {
let builder = AdaptyProfileParameters.Builder() .with(firebaseAppInstanceId: appInstanceId)

Adapty.updateProfile(params: builder.build()) { error in
            // handle error
}

} </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin //after Adapty.activate()

FirebaseAnalytics.getInstance(context).appInstanceId.addOnSuccessListener { appInstanceId -> Adapty.updateProfile( AdaptyProfileParameters.Builder() .withFirebaseAppInstanceId(appInstanceId) .build() ) { //handle error } } </TabItem> <TabItem value="java" label="Java" default> java //after Adapty.activate()

FirebaseAnalytics.getInstance(context).getAppInstanceId().addOnSuccessListener(appInstanceId -> { AdaptyProfileParameters params = new AdaptyProfileParameters.Builder()
.withFirebaseAppInstanceId(appInstanceId) .build();

Adapty.updateProfile(params, error -> {
    if (error != null) {
        // handle the error
    }
});

}); </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:firebase*analytics/firebase*analytics.dart';

final builder = AdaptyProfileParametersBuilder() ..setFirebaseAppInstanceId( await FirebaseAnalytics.instance.appInstanceId, );

try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp // We suppose FirebaseAnalytics Unity Plugin is already installed

Firebase.Analytics .FirebaseAnalytics .GetAnalyticsInstanceIdAsync() .ContinueWithOnMainThread((task) => { if (!task.IsCompletedSuccessfully) { // handle error return; }

            var firebaseId = task.Result
            var builder = new Adapty.ProfileParameters.Builder();
            builder.SetFirebaseAppInstanceId(firebaseId);

            Adapty.UpdateProfile(builder.Build(), (error) => {
                // handle error
            });
        });

</TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import analytics from '@react-native-firebase/analytics'; import { adapty } from 'react-native-adapty';

try { const appInstanceId = await analytics().getAppInstanceId();

await adapty.updateProfile({ firebaseAppInstanceId: appInstanceId, }); } catch (error) { // handle `AdaptyError` } ```

## Sending events and user properties

And now it is time to decide which events you will receive in Firebase and Google Analytics.

upevents_names.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can see that some events have designated names, for example. "Purchase", while other ones are usual Adapty events. This discrepancy comes from Google Analytics event types. Currently, supported events are Refund and Purchase. Other events are custom events. So, please ensure that your event names are supported by Google Analytics.
Also, you can set up sending user properties in the Adapty dashboard.

2023-08-21at_12.50.162x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

This means that your events will be enriched with `subscription_state` and `subscription_product_id` by Adapty. But you also have to enable this feature in Google Analytics. So to use**User properties** in your analytics, begin by assigning them to a custom dimension through the Firebase Console's **Custom Definitions** by selecting the **User scope**, naming, and describing them.

2023-08-21at_12.48.222x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2023-08-21at_12.52.532x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Please check that your user property names are `subscription_state` and `subscription_product_id`. Otherwise, we won't be able to send you subscription status data.

:::note There is a time delay between when events are sent from Adapty and when they appear on the Google Analytics Dashboard. It's suggested to monitor the Realtime Dashboard on your Google Analytics account to see the latest events in real-time. :::

And that's all! Wait for new insights from Google.
```

title: "Mixpanel" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Mixpanel is a powerful product analytics service. Its event-driven tracking solution empowers product teams to get valuable insights into optimal user acquisition, conversion, and retention strategies across different platforms.

This integration enables you to bring all the Adapty events into Mixpanel. As a result, you'll gain a more comprehensive insight into your subscription business and customer actions. Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

How to set up Mixpanel integration

To set up the integration with Mixpanel, go to **Integrations -> Mixpanel** in the Adapty Dashboard, turn on a toggle, and fill out fields.

2023-08-17at_14.21.392x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You need only one variable such as **Mixpanel token**. You can find the token in your Mixpanel project. If you need help, here's the official docs.

2023-08-16at_18.09.382x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty maps some properties such as user id and revenue of the event to Mixpanel-native properties. With such mapping, Mixpanel is able to show you the correct data in the profile and events timeline.

Adapty also accumulates revenue from each user.

Another thing worth mentioning is updating User Profile Properties. Adapty sends the `subscription state` and `subscription product id`. After Mixpanel gets an event, you can see the corresponding fields updated there.

Events and tags

Below the credentials, there are three groups of events you can send to Mixpanel from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-11at_14.56.362x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

SDK configuration

Use `Adapty.updateProfile()` method to set `mixpanelUserId`. If not set, Adapty uses your user ID (`customerUserId`) or if it's null Adapty ID. Make sure that the user id you use to send data to Mixpanel from your app is the same one you send to Adapty.

```swift import Mixpanel

let builder = AdaptyProfileParameters.Builder() .with(mixpanelUserId: Mixpanel.mainInstance().distinctId)

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withMixpanelUserId(mixpanelAPI.distinctId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:mixpanel_flutter/mixpanel_flutter.dart';

final mixpanel = await Mixpanel.init("Your Token", trackAutomaticEvents: true);

final builder = AdaptyProfileParametersBuilder() ..setMixpanelUserId( await mixpanel.getDistinctId(), );

```
try {
    await Adapty().updateProfile(builder.build());
} on AdaptyError catch (adaptyError) {
    // handle error
} catch (e) {}
```

</TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetMixpanelUserId(Mixpanel.DistinctId);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import { Mixpanel } from 'mixpanel-react-native';

// ... try { await adapty.updateProfile({ mixpanelUserId: mixpanelUserId, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "Mixpanel" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Mixpanel is a powerful product analytics service. Its event-driven tracking solution empowers product teams to get valuable insights into optimal user acquisition, conversion, and retention strategies across different platforms.

This integration enables you to bring all the Adapty events into Mixpanel. As a result, you'll gain a more comprehensive insight into your subscription business and customer actions. Adapty provides a complete set of data that lets you track subscription events from stores in one place. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective.

## How to set up Mixpanel integration

To set up the integration with Mixpanel, go to **Integrations -> Mixpanel** in the Adapty Dashboard, turn on a toggle, and fill out fields.

2023-08-17at_14.21.392x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You need only one variable such as **Mixpanel token**. You can find the token in your Mixpanel project. If you need help, here's the official docs.

2023-08-16at_18.09.382x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty maps some properties such as user id and revenue of the event to Mixpanel-native properties. With such mapping, Mixpanel is able to show you the correct data in the profile and events timeline.

Adapty also accumulates revenue from each user.

Another thing worth mentioning is updating User Profile Properties. Adapty sends the `subscription state` and `subscription product id`. After Mixpanel gets an event, you can see the corresponding fields updated there.

## Events and tags

Below the credentials, there are three groups of events you can send to Mixpanel from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-11at_14.56.362x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs.

## SDK configuration

Use `Adapty.updateProfile()` method to set `mixpanelUserId`. If not set, Adapty uses your user ID (`customerUserId`) or if it's null Adapty ID. Make sure that the user id you use to send data to Mixpanel from your app is the same one you send to Adapty.

```swift import Mixpanel

let builder = AdaptyProfileParameters.Builder() .with(mixpanelUserId: Mixpanel.mainInstance().distinctId)

Adapty.updateProfile(params: builder.build()) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withMixpanelUserId(mixpanelAPI.distinctId) .build() Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:mixpanel*flutter/mixpanel*flutter.dart';

final mixpanel = await Mixpanel.init("Your Token", trackAutomaticEvents: true);

final builder = AdaptyProfileParametersBuilder() ..setMixpanelUserId( await mixpanel.getDistinctId(), );

```
try {
    await Adapty().updateProfile(builder.build());
} on AdaptyError catch (adaptyError) {
    // handle error
} catch (e) {}
```

</TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetMixpanelUserId(Mixpanel.DistinctId);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import { Mixpanel } from 'mixpanel-react-native';

// ... try { await adapty.updateProfile({ mixpanelUserId: mixpanelUserId, }); } catch (error) { // handle AdaptyError } ```

---

title: "SplitMetrics Acquire" description: ""

metadataTitle: ""

With SplitMetrics Acquire integration, you can see exactly how much money your Apple Search Ads make from subscriptions. And you can track your users for months to know how much money your ads make over time.

In addition, Adapty sends subscription events to SplitMetrics Acquire so that you can build custom dashboards and automation there, based on Apple Search Ads attribution.
It doesn't add any attribution data to Adapty, as we already have everything we need from ASA directly.

How to set up SplitMetrics Acquire integration

To integrate SplitMetrics Acquire go to Integrations > SplitMetrics Acquire and set credentials.

2023-08-14at_17.39.422x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Open your SplitMetrics Acquire account, hover over one of the MMP logos and click the appeared **Settings** button. Find your Client ID in the dialog under item 5, copy it, and then paste it to Adapty as Client ID.

./ width: '700px', image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

./ width: '700px', image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

You will also have to set Apple App ID to use the integration. To find App ID, open your app page in App Store Connect, go to the App Information page in section General and find Apple ID in the left bottom part of the screen.

2022-04-20at_17.55.03.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Events and tags

Below the credentials, there are three groups of events you can send to SplitMetrics Acquire from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-11at_14.56.362x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

We recommend using the default event names provided by Adapty. But you can change the event names based on your needs. Adapty will send subscription events to SplitMetrics Acquire using a server-to-server integration, allowing you to view all subscription events in your SplitMetrics dashboard.

SDK configuration

You don't have to configure anything on the SDK side, but we recommend sending `customerUserId` to Adapty for better accuracy.

:::warning Make sure you've configured Apple Search Ads in Adapty and uploaded credentials, without them, SplitMetrics Acquire won't work. :::

---

title: "Slack" description: ""

metadataTitle: ""

Slack is a workplace messenger and productivity platform that probably needs no introduction.

With this integration, you'll be able to be notified in Slack each time a revenue event is tracked by Adapty. This can be helpful if you like to cherish every moment your MRR increases or if you'd like to be on the lookout for trial cancellations, billing issues, refunds, and more.

How to set up Slack integration

You'll need to:

- create an app in your Slack workspace
- give it permission to post messages
- and then provide the necessary info to Adapty in Integrations â†’ Slack.

1. Create an app in Slack

Go to Slack API dashboard and create an app like so:

2024-01-04at_18.27.412x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

2024-01-04at_18.28.142x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Give it any name ("Adapty" for example) and add it to your workspace:

2024-01-04at_18.29.132x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

2. Give permission to post and get a token for your app

You'll be redirected to your app's page in Slack. Scroll down and press Permissions:

2024-01-04at_18.48.072x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

After the redirect, scroll down to Scopes and press "Add an OAuth Scope":

2024-01-04at_18.50.262x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Give `chat:write`, `chat:write.public` and `chat:write.customize` permissions. Those are needed to post in your channels and customize the messages:

2024-01-04at_18.51.572x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Scroll back to the top of the page and press "Install to Workspace":

2024-01-04at_19.17.58.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Press "Allow" here:

2024-01-04at_18.53.292x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

After this, you'll be redirected to the same page, but you'll have an OAuth Token available (`xoxb-...`). This is exactly what's needed to complete the setup:

3. Configure the integration in Adapty

Go to **Integrations â†' Slack**:

Paste the `xoxb-...` token from the previous step and choose which channels the app will post to. You can set up the integration to receive events only on production, sandbox or both. You can also choose which currency to post in (original or converted to USD).

:::note Note that if you'd like to post messages from Adapty in a private channel, you'll need to manually add the "Adapty" app you've created in Slack to that channel. Otherwise it would not work. :::

Finally, you can choose which events you'd like to receive under "Events":

:::info You're all set!

The events will be sent to the channels you've specified. You'll be able to see the revenue where applicable and view the customer profile in Adapty: :::

---

title: "Slack" description: ""

metadataTitle: ""

Slack is a workplace messenger and productivity platform that probably needs no introduction.

With this integration, you'll be able to be notified in Slack each time a revenue event is tracked by Adapty. This can be helpful if you like to cherish every moment your MRR increases or if you'd like to be on the lookout for trial cancellations, billing issues, refunds, and more.

How to set up Slack integration

You'll need to:

- create an app in your Slack workspace
- give it permission to post messages
- and then provide the necessary info to Adapty in Integrations â†' Slack.

1. Create an app in Slack

Go to Slack API dashboard and create an app like so:

Give it any name ("Adapty" for example) and add it to your workspace:

2. Give permission to post and get a token for your app

You'll be redirected to your app's page in Slack. Scroll down and press Permissions:

After the redirect, scroll down to Scopes and press "Add an OAuth Scope":

Give `chat:write`, `chat:write.public` and `chat:write.customize` permissions. Those are needed to post in your channels and customize the messages:

Scroll back to the top of the page and press "Install to Workspace":

Press "Allow" here:

After this, you'll be redirected to the same page, but you'll have an OAuth Token available (`xoxb-...`). This is exactly what's needed to complete the setup:

3. Configure the integration in Adapty

Go to **Integrations â†' Slack**:

Paste the `xoxb-...` token from the previous step and choose which channels the app will post to. You can set up the integration to receive events only on production, sandbox or both. You can also choose which currency to post in (original or converted to USD).

:::note Note that if you'd like to post messages from Adapty in a private channel, you'll need to manually add the "Adapty" app you've created in Slack to that channel. Otherwise it would not work. :::

Finally, you can choose which events you'd like to receive under "Events":

:::info You're all set!

The events will be sent to the channels you've specified. You'll be able to see the revenue where applicable and view the customer profile in Adapty: :::

---

title: "Messaging service integrations" description: ""

metadataTitle: ""

The acquisition is not easy or cheap in the growing mobile market. So wisely treating attracted users improves your unit economy, especially in highly competitive niches.

Adapty provides real-time information about core users' payment actions. We know when your customer took a trial, if he had troubles with his payment, or if he purchased a subscription and decided to cancel later. All these and other events show the change in the state of the customer. And this is the best moment to react - send an offer, or personal gift, or whatever retaining.

Push notification platforms allow describing a user with standard and custom tags to build an effective automatic system of retention. To make this system work you just need trigger events to let the system know that it's time to send a message. These events will come to the push platform from Adapty through the set integration.

Please choose below the service that you need to integrate and follow the instructions:

- Braze
- OneSignal
- Pushwoosh

:::note Don't see your attribution provider?

Let us know! Write to the Adapty support and we'll consider adding it. :::

---

title: "Braze" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

As one of the top customer engagement solutions, Braze provides a wide range of tools for push notifications, email, SMS, and in-app messaging. By integrating Adapty with Braze, you can easily access all of your subscription events in one place, giving you the ability to trigger automated communication based on those events.

Adapty provides a complete set of data that lets you track subscription events from all stores in one place and can be used to update your users' profiles in Braze. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in your Braze dashboard and map them with your acquisition campaigns.

Adapty sends subscription events, user properties and purchases over to Braze, so you can build target communication with customers using Braze push notifications after a short and easy integration as described below.

How to set up Braze integration

To integrate Braze go to **Integrations -> Braze**, switch on the toggle, and fill out the fields.

The initial step of the integration process is to provide the necessary credentials to establish a connection between your Braze and Adapty profiles. You will need the **REST API Key**, your **Braze Instance ID**, and **App IDs** for iOS and Android for the integration to work properly:

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

1. **REST API Key** can be created in **Braze Dashboard** â†’ **Settings** â†’ **API Keys**. Make sure your key has a `users.track` permission when creating it:

./brazecreate*api*key.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

./brazeapi*key*users_track.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

2. To get **Braze Instance ID** note your Braze Dashboard URL and go to the section of Braze Docs where the instance ID is specified. It should have a regional form such as US-03, EU-01, etc.
3. iOS and Android App IDs can be found in Braze Dashboard â†’ Settings â†’ API Keys as well. Copy them from here:

./brazeapp_ids.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Events, user attributes and purchases

Below the credentials, there are three groups of events you can send to Braze from Adapty. Simply turn on the ones you need. You may also change the names of the events as you need to send it to Braze. Check the full list of the Events offered by Adapty here:

./brazeevents_names.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Adapty will send subscription events and user attributes to Braze using a server-to-server integration, allowing you to view it in your Braze Dashboard and configure campaigns based on that.

For events that have revenue, such as trial conversions and renewals, Adapty will send this info to Braze as purchases.

Here you can find the complete specifications for the event properties sent to Braze.

:::note Helpful user attributes

Adapty sends some user attributes for Braze integration by default. You can refer to the list of them provided below to determine which is best suited for your needs. :::

| User attribute | Type | Value | |--------------|----|-----| `adapty_customer_user_id` | String | Contains the value of the unique identifier of the user defined by the customer. Can be found both in the Adapty Dashboard and in Braze. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in the Adapty Dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product Id in Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

YYYY-MM-DDTHH:mm:ss.SSS+TZ

e.g., 2023-02-15T17:22:03.000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for a trial period or `normal` for the rest.

|

All float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to Braze if the user marks the ** Send user attributes** checkbox from the integration page

SDK Configuration

To link user profiles in Adapty and Braze you need to either configure Braze SDK with the same customer user ID as Adapty or use its `.changeUser()` method:

```
swift let braze = Braze(configuration: configuration) braze.changeUser(userId: "adapty_customer_user_id") kotlin Braze.getInstance(context).changeUser("adapty_customer_user_id")
```

---

title: "Braze" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

As one of the top customer engagement solutions, Braze provides a wide range of tools for push notifications, email, SMS, and in-app messaging. By integrating Adapty with Braze, you can easily access all of your subscription events in one place, giving you the ability to trigger automated communication based on those events.

Adapty provides a complete set of data that lets you track subscription events from all stores in one place and can be used to update your users' profiles in Braze. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in your Braze dashboard and map them with your acquisition campaigns.

Adapty sends subscription events, user properties and purchases over to Braze, so you can build target communication with customers using Braze push notifications after a short and easy integration as described below.

How to set up Braze integration

To integrate Braze go to **Integrations -> Braze**, switch on the toggle, and fill out the fields.

The initial step of the integration process is to provide the necessary credentials to establish a connection between your Braze and Adapty profiles. You will need the **REST API Key**, your **Braze Instance ID**, and **App IDs** for iOS and Android for the integration to work properly:

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

1. **REST API Key** can be created in **Braze Dashboard** â†’ **Settings** â†’ **API Keys**. Make sure your key has a `users.track` permission when creating it:

./brazecreate*api*key.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

./brazeapi*key*users_track.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

2. To get **Braze Instance ID** note your Braze Dashboard URL and go to the section of Braze Docs where the instance ID is specified. It should have a regional form such as US-03, EU-01, etc.
3. iOS and Android App IDs can be found in Braze Dashboard â†’ Settings â†’ API Keys as well. Copy them from here:

./brazeapp_ids.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Events, user attributes and purchases

Below the credentials, there are three groups of events you can send to Braze from Adapty. Simply turn on the ones you need. You may also change the names of the events as you need to send it to Braze. Check the full list of the Events offered by Adapty here:

./brazeevents_names.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Adapty will send subscription events and user attributes to Braze using a server-to-server integration, allowing you to view it in your Braze Dashboard and configure campaigns based on that.

For events that have revenue, such as trial conversions and renewals, Adapty will send this info to Braze as purchases.

Here you can find the complete specifications for the event properties sent to Braze.

:::note Helpful user attributes

Adapty sends some user attributes for Braze integration by default. You can refer to the list of them provided below to determine which is best suited for your needs. :::

| User attribute | Type | Value | |----------|----|-----| `adapty_customer_user_id` | String | Contains the value of the unique identifier of the user defined by the customer. Can be found both in the Adapty Dashboard and in Braze. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in the Adapty Dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product Id in Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

YYYY-MM-DDTHH:mm:ss.SSS+TZ

e.g., 2023-02-15T17:22:03.000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for a trial period or `normal` for the rest.

|

All float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to Braze if the user marks the ** Send user attributes** checkbox from the integration page

SDK Configuration

To link user profiles in Adapty and Braze you need to either configure Braze SDK with the same customer user ID as Adapty or use its `.changeUser()` method:

```
swift let braze = Braze(configuration: configuration) braze.changeUser(userId: "adapty_customer_user_id") kotlin Braze.getInstance(context).changeUser("adapty_customer_user_id")
```

---

title: "OneSignal" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

As one of the top customer engagement solutions, OneSignal provides a wide range of tools for push notifications, email, SMS, and in-app messaging. By integrating Adapty with OneSignal, you can easily access all of your subscription events in one place, giving you the ability to trigger automated communication based on those events.

Adapty provides a complete set of data that lets you track subscription events from all stores in one place and can be used to update your OneSignal users. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in your OneSignal dashboard and map them with your acquisition campaigns.

Adapty uses subscription events to update OneSignal tags, so you can build target communication with customers using OneSignal push notifications after a short and easy integration setting as described below.

How to set up One Signal integration

To set up the integration with OneSignal, go to **Integrations -> OneSignal** in your Adapty dashboard, turn on a toggle from off to on, and fill out fields.

Set up credentials in the Adapty Dashboard

The initial step of the integration process is to provide the necessary credentials to establish a connection between your OneSignal and Adapty profiles.
You'll need to provide your **OneSignal App ID** and **Auth Token**. You can find more information about OneSignal Keys and IDs in following documentation.

Retrieving credentials from OneSignal dashboard

To find your OneSignal app ID and authentication key, simply navigate to your OneSignal dashboard.
Your **App ID** can be found under the **Keys & IDs* *section in the Settings tab.

The Auth token can be found in the **Account & API Keys** section of your OneSignal dashboard.

After retrieving your OneSignal App ID and authentication key from the OneSignal dashboard, you need to add them to the Adapty dashboard in the corresponding fields.

Events and tags

Below the credentials, there are three groups of events you can send to OneSignal from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

Adapty will send subscription events to OneSignal using a server-to-server integration, allowing you to view all subscription events in your OneSignal Dashboard and link them to your acquisition campaigns.

:::warning Please consider that starting from April 17th, 2023, it will not be possible to send attribution data from Adapty to OneSignal, if you are using the Free Plan of OneSignal. This integration is only available for OneSignal's Growth, Professional and higher plans. For more info please check OneSignal pricing.

Furthermore, it's important to note that the tag limitation applies to all the tags you have set up in OneSignal, including any existing tags. When sending data from Adapty to OneSignal, the tag limit in OneSignal includes both the tags sent from Adapty and the tags you may have already defined in OneSignal. Therefore, if you exceed the tag limit in OneSignal, it may result in errors when sending events from Adapty. :::

:::note Custom tags

This integration can update and set various properties in your Adapty users as tags that will be send to OneSignal. You can refer to the list of tags provided below to determine which tag is best suited for your needs. :::

| Tag | Type | Description | |---|----|----------| |`adapty_customer_user_id` | String | Contains the value of the unique identifier of the user, which can be found from OneSignal side. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in your Adapty dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product Id in Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `last_event_type` | String | Indicates the type of the last received event from the list of the standard Adapty events that you have enabled for the integration. | | `purchase_date` | String |

Contains the date of the last transaction (original purchase or renewal).

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for trial period or `normal` for the rest.

|

Please consider that all float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to OneSignal if the user marks the * *Send User Attributes** checkbox from the integration page. When unchecked, Adapty sends exactly 10 tags. If the checkbox is checked, we can send more than 10 tags for greater flexibility in capturing relevant data.

SDK configuration

There are currently two ways to integrate OneSignal and Adapty: the old one, relying on `playerId` and the new one, relying on `subscriptionId`, since `playerId` is deprecated starting with OneSignal SDK v5

:::warning Make sure you send `playerId` (on OneSignal SDK pre-v5) or `subscriptionId` (on OneSignal SDK v5+) to Adapty, otherwise OneSignal tags couldn't be updated and the integration wouldn't work. :::

Here is how you can link Adapty with OneSignal with either `playerId` or `subscriptionId`:

```swift // PlayerID (pre-v5 OneSignal SDK) // in your OSSubscriptionObserver implementation func onOSSubscriptionChanged(_ stateChanges: OSSubscriptionStateChanges) { if let playerId = stateChanges.to.userId { let params = AdaptyProfileParameters.Builder() .with(oneSignalPlayerId: playerId) .build()

      Adapty.updateProfile(params:params) { error in
          // check error
      }
}

}

// SubscriptionID (v5+ OneSignal SDK) OneSignal.Notifications.requestPermission({ accepted in let id = OneSignal.User.pushSubscription.id

let builder = AdaptyProfileParameters.Builder()
    .with(oneSignalSubscriptionId: id)

Adapty.updateProfile(params: builder.build())
```

}, fallbackToSettings: true) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin // PlayerID (pre-v5 OneSignal SDK) val osSubscriptionObserver = OSSubscriptionObserver { stateChanges -> stateChanges?.to?.userId?.let { playerId -> val params = AdaptyProfileParameters.Builder() .withOneSignalPlayerId(playerId) .build()

```
    Adapty.updateProfile(params) { error ->
        if (error != null) {
            // handle the error
        }
    }
}
```

}

// SubscriptionID (v5+ OneSignal SDK) val oneSignalSubscriptionObserver = object: IPushSubscriptionObserver { override fun onPushSubscriptionChange(state: PushSubscriptionChangedState) { val params = AdaptyProfileParameters.Builder() .withOneSignalSubscriptionId(state.current.id) .build()

```
    Adapty.updateProfile(params) { error ->
        if (error != null) {
            // handle the error
        }
    }
}
```

} </TabItem> <TabItem value="java" label="Java" default> java // PlayerID (pre-v5 OneSignal SDK) OSSubscriptionObserver osSubscriptionObserver = stateChanges -> { OSSubscriptionState to = stateChanges != null ? stateChanges.getTo() : null; String playerId = to != null ? to.getUserId() : null;

```
if (playerId != null) {
    AdaptyProfileParameters params1 = new AdaptyProfileParameters.Builder()
            .withOneSignalPlayerId(playerId)
            .build();

    Adapty.updateProfile(params1, error -> {
        if (error != null) {
            // handle the error
        }
    });
}
```

};

// SubscriptionID (v5+ OneSignal SDK) IPushSubscriptionObserver oneSignalSubscriptionObserver = state -> { AdaptyProfileParameters params = new AdaptyProfileParameters.Builder() .withOneSignalSubscriptionId(state.getCurrent().getId()) .build(); Adapty.updateProfile(params, error -> { if (error != null) { // handle the error } }); }; </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript OneSignal.shared.setSubscriptionObserver((changes) { final playerId = changes.to.userId; if (playerId != null) { final builder = AdaptyProfileParametersBuilder() ..setOneSignalPlayerId(playerId); try { Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) { // handle error } } }); </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp using OneSignalSDK;

var pushUserId = OneSignal.Default.PushSubscriptionState.userId;

var builder = new Adapty.ProfileParameters.Builder(); builder.SetOneSignalPlayerId(pushUserId);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import OneSignal from 'react-native-onesignal';

OneSignal.addSubscriptionObserver(event => { const playerId = event.to.userId;

adapty.updateProfile({ oneSignalPlayerId: playerId, }); }); ```

Read more about `OSSubscriptionObserver` in OneSignal documentation.

Dealing with multiple devices

One can often encounter the following situation: a single user has different devices and analytics of purchase events or user subscriptions becomes difficult. OneSignal suggests methods to cope with this problem. You can match different devices on your server side and send this information to OneSignal. Thus, when you change the user's tags, they will be updated not only for a specific device but for all devices the user has.

To take advantage of this feature, Adapty provides the ability to identify users using the Adapty SDK and send their ID to OneSignal. By leveraging this opportunity, you can easily match a user's devices and update tags for multiple devices without any extra actions. This not only simplifies the process of tracking user activity and subscriptions but also allows you to provide a seamless experience for your users across all their devices.

It's important to note that to properly match a user's devices, you must ensure that each user has a unique identifier. Adapty's `customer_user_id` can be used as an `externalUserId` for this purpose, but if your app doesn't have a registration system, you may need to use a different identifier. Additionally, it's crucial to keep this identifier consistent across all devices and to send updates to OneSignal whenever a user's ID changes. By keeping track of their activity and subscriptions across all devices, you can better understand their needs and provide them with relevant and timely notifications.

---

title: "OneSignal" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

As one of the top customer engagement solutions, OneSignal provides a wide range of tools for push notifications, email, SMS, and in-app messaging. By integrating Adapty with OneSignal, you can easily access all of your subscription events in one place, giving you the ability to trigger automated communication based on those events.

Adapty provides a complete set of data that lets you track subscription events from all stores in one place and can be used to update your OneSignal users. With Adapty, you can easily see how your subscribers are behaving, learn what they like, and use that information to communicate with them in a way that's targeted and effective. Therefore, this integration allows you to track subscription events in your OneSignal dashboard and map them with your acquisition campaigns.

Adapty uses subscription events to update OneSignal tags, so you can build target communication with customers using OneSignal push notifications after a short and easy integration setting as described below.

How to set up One Signal integration

To set up the integration with OneSignal, go to **Integrations -> OneSignal** in your Adapty dashboard, turn on a toggle from off to on, and fill out fields.

Set up credentials in the Adapty Dashboard

The initial step of the integration process is to provide the necessary credentials to establish a connection between your OneSignal and Adapty profiles.
You'll need to provide your **OneSignal App ID** and **Auth Token**. You can find more information about OneSignal Keys and IDs in following documentation.

2023-08-17at_15.07.162x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Retrieving credentials from OneSignal dashboard

To find your OneSignal app ID and authentication key, simply navigate to your OneSignal dashboard.
Your **App ID** can be found under the *Keys & IDs *section in the Settings tab.

2023-08-17at_15.10.262x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

The Auth token can be found in the **Account & API Keys** section of your OneSignal dashboard.

2023-08-17at15.14.532x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', */ image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

After retrieving your OneSignal App ID and authentication key from the OneSignal dashboard, you need to add them to the Adapty dashboard in the corresponding fields.

Events and tags

Below the credentials, there are three groups of events you can send to OneSignal from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

/ width: '700px', / image width */ display: 'block', */ for alignment */ margin: '0 auto' */ center alignment */ }} />

Adapty will send subscription events to OneSignal using a server-to-server integration, allowing you to view all subscription events in your OneSignal Dashboard and link them to your acquisition campaigns.

:::warning Please consider that starting from April 17th, 2023, it will not be possible to send attribution data from Adapty to OneSignal, if you are using the Free Plan of OneSignal. This integration is only available for OneSignal's Growth, Professional and higher plans. For more info please check OneSignal pricing.

Furthermore, it's important to note that the tag limitation applies to all the tags you have set up in OneSignal, including any existing tags. When sending data from Adapty to OneSignal, the tag limit in OneSignal includes both the tags sent from Adapty and the tags you may have already defined in OneSignal. Therefore, if you exceed the tag limit in OneSignal, it may result in errors when sending events from Adapty. :::

:::note Custom tags

This integration can update and set various properties in your Adapty users as tags that will be send to OneSignal. You can refer to the list of tags provided below to determine which tag is best suited for your needs. :::

| Tag | Type | Description | |---|----|----------| |`adapty_customer_user_id` | String | Contains the value of the unique identifier of the user, which can be found from OneSignal side. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in your Adapty dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product Id in Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `last_event_type` | String | Indicates the type of the last received event from the list of the standard Adapty events that you have enabled for the integration. | | `purchase_date` | String |

Contains the date of the last transaction (original purchase or renewal).

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for trial period or `normal` for the rest.

|

Please consider that all float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to OneSignal if the user marks the *** Send User Attributes*** checkbox from the integration page. When unchecked, Adapty sends exactly 10 tags. If the checkbox is checked, we can send more than 10 tags for greater flexibility in capturing relevant data.

SDK configuration

There are currently two ways to integrate OneSignal and Adapty: the old one, relying on `playerId` and the new one, relying on `subscriptionId`, since `playerId` is deprecated starting with OneSignal SDK v5

:::warning Make sure you send `playerId` (on OneSignal SDK pre-v5) or `subscriptionId` (on OneSignal SDK v5+) to Adapty, otherwise OneSignal tags couldn't be updated and the integration wouldn't work. :::

Here is how you can link Adapty with OneSignal with either `playerId` or `subscriptionId`:

```swift // PlayerID (pre-v5 OneSignal SDK) // in your OSSubscriptionObserver implementation func onOSSubscriptionChanged(_ stateChanges: OSSubscriptionStateChanges) { if let playerId = stateChanges.to.userId { let params = AdaptyProfileParameters.Builder() .with(oneSignalPlayerId: playerId) .build()

```
      Adapty.updateProfile(params:params) { error in
          // check error
      }
}
```

}

// SubscriptionID (v5+ OneSignal SDK) OneSignal.Notifications.requestPermission({ accepted in let id = OneSignal.User.pushSubscription.id

```
let builder = AdaptyProfileParameters.Builder()
    .with(oneSignalSubscriptionId: id)

Adapty.updateProfile(params: builder.build())
```

}, fallbackToSettings: true) </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin // PlayerID (pre-v5 OneSignal SDK) val osSubscriptionObserver = OSSubscriptionObserver { stateChanges -> stateChanges?.to?.userId?.let { playerId -> val params = AdaptyProfileParameters.Builder() .withOneSignalPlayerId(playerId) .build()

```
    Adapty.updateProfile(params) { error ->
        if (error != null) {
            // handle the error
        }
    }
}
```

}

// SubscriptionID (v5+ OneSignal SDK) val oneSignalSubscriptionObserver = object: IPushSubscriptionObserver { override fun onPushSubscriptionChange(state: PushSubscriptionChangedState) { val params = AdaptyProfileParameters.Builder() .withOneSignalSubscriptionId(state.current.id) .build()

```
    Adapty.updateProfile(params) { error ->
        if (error != null) {
            // handle the error
        }
    }
}
```

} </TabItem> <TabItem value="java" label="Java" default> java // PlayerID (pre-v5 OneSignal SDK) OSSubscriptionObserver osSubscriptionObserver = stateChanges -> { OSSubscriptionState to = stateChanges != null ? stateChanges.getTo() : null; String playerId = to != null ? to.getUserId() : null;

```
if (playerId != null) {
    AdaptyProfileParameters params1 = new AdaptyProfileParameters.Builder()
            .withOneSignalPlayerId(playerId)
            .build();

    Adapty.updateProfile(params1, error -> {
        if (error != null) {
            // handle the error
        }
    });
}
```
```

};

// SubscriptionID (v5+ OneSignal SDK) IPushSubscriptionObserver oneSignalSubscriptionObserver = state -> { AdaptyProfileParameters params = new AdaptyProfileParameters.Builder() .withOneSignalSubscriptionId(state.getCurrent().getId()) .build(); Adapty.updateProfile(params, error -> { if (error != null) { // handle the error } }); }; `</TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default>` javascript OneSignal.shared.setSubscriptionObserver((changes) { final playerId = changes.to.userId; if (playerId != null) { final builder = AdaptyProfileParametersBuilder() ..setOneSignalPlayerId(playerId); try { Adapty().updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) { // handle error } } }); `</TabItem> <TabItem value="Unity" label="Unity (C#)" default>` csharp using OneSignalSDK;

var pushUserId = OneSignal.Default.PushSubscriptionState.userId;

var builder = new Adapty.ProfileParameters.Builder(); builder.SetOneSignalPlayerId(pushUserId);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript import { adapty } from 'react-native-adapty'; import OneSignal from 'react-native-onesignal';

OneSignal.addSubscriptionObserver(event => { const playerId = event.to.userId;

adapty.updateProfile({ oneSignalPlayerId: playerId, }); }); ```

Read more about `OSSubscriptionObserver` in OneSignal documentation.

Dealing with multiple devices

One can often encounter the following situation: a single user has different devices and analytics of purchase events or user subscriptions becomes difficult. OneSignal suggests methods to cope with this problem. You can match different devices on your server side and send this information to OneSignal. Thus, when you change the user's tags, they will be updated not only for a specific device but for all devices the user has.

To take advantage of this feature, Adapty provides the ability to identify users using the Adapty SDK and send their ID to OneSignal. By leveraging this opportunity, you can easily match a user's devices and update tags for multiple devices without any extra actions. This not only simplifies the process of tracking user activity and subscriptions but also allows you to provide a seamless experience for your users across all their devices.

It's important to note that to properly match a user's devices, you must ensure that each user has a unique identifier. Adapty's `customer_user_id` can be used as an `externalUserId` for this purpose, but if your app doesn't have a registration system, you may need to use a different identifier. Additionally, it's crucial to keep this identifier consistent across all devices and to send updates to OneSignal whenever a user's ID changes. By keeping track of their activity and subscriptions across all devices, you can better understand their needs and provide them with relevant and timely notifications.

---

title: "Pushwoosh" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty uses subscription events to update Pushwoosh profile tags, so you can build target communication with customers using push notifications after a short and easy integration setting as described below.

How to set up Pushwoosh integration

To integrate Pushwoosh go to **Integrations -> Pushwoosh**, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between your Pushwoosh and Adapty profiles.
Pushwoosh app ID and auth token are required.

2023-08-18at_11.13.212x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

1. **App ID** can be found in your Pushwoosh dashboard.

2023-08-18at_14.37.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. ***Auth token** *can be found in the API Access section in Pushwoosh Settings.

2023-08-18at_14.35.022x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Events and tags

Below the credentials, there are three groups of events you can send to Pushwoosh from Adapty. Simply turn on the ones you need. You may also change the names of the events as you need to send it to Pushwoosh. Check the full list of the Events offered by Adapty here.

3107.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty will send subscription events to Pushwoosh using a server-to-server integration, allowing you to view all subscription events in your Pushwoosh Dashboard.

:::note Custom tags

With Adapty you can also use your custom tags for Pushwoosh integration. You can refer to the list of tags provided below to determine which tag is best suited for your needs. :::

| Tag | Type | Value | |---|----|-----| | `adapty_customer_user_id` | String | Contains the value of the unique identifier of the user, which can be found on the Pushwoosh side. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in your Adapty dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product ID in the Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `last_event_type` | String | Indicates the type of the last received event from the list of the standard Adapty events that you have enabled for the integration. | | `purchase_date` | String |

Contains the date of the last transaction (original purchase or renewal).

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `original_purchase_date` | String |

Contains the date of the first purchase according to the transaction.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for a trial period or `normal` for the rest.

|

All float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to Pushwoosh if the user marks the *** Send user custom attributes*** checkbox from the integration page

SDK configuration

To link Adapty with Pushwoosh, you need to send us the `HWID` value:

```swift let params = AdaptyProfileParameters.Builder() .with(pushwooshHWID: Pushwoosh.sharedInstance().getHWID()) .build()

Adapty.updateProfile(params: params) { error in // handle the error } </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withPushwooshHwid(Pushwoosh.getInstance().hwid) .build()

Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="java" label="Java" default> java AdaptyProfileParameters params = new AdaptyProfileParameters.Builder() .withPushwooshHwid(Pushwoosh.getInstance().getHwid()) .build();

Adapty.updateProfile(params, error -> { if (error != null) { // handle the error } }); </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:pushwoosh/pushwoosh.dart';

final builder = AdaptyProfileParametersBuilder() ..setPushwooshHWID( await Pushwoosh.getInstance.getHWID, ); try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetPushwooshHWID(Pushwoosh.Instance.HWID);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import Pushwoosh from 'pushwoosh-react-native-plugin';

// ... try { await adapty.updateProfile({ pushwooshHWID: hwid, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "Pushwoosh" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty uses subscription events to update Pushwoosh profile tags, so you can build target communication with customers using push notifications after a short and easy integration setting as described below.

## How to set up Pushwoosh integration

To integrate Pushwoosh go to **Integrations -> Pushwoosh**, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between your Pushwoosh and Adapty profiles.
Pushwoosh app ID and auth token are required.

2023-08-18at_11.13.212x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

1. **App ID** can be found in your Pushwoosh dashboard.

2023-08-18at_14.37.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. ***Auth token** *can be found in the API Access section in Pushwoosh Settings.

2023-08-18at_14.35.022x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

## Events and tags

Below the credentials, there are three groups of events you can send to Pushwoosh from Adapty. Simply turn on the ones you need. You may also change the names of the events as you need to send it to Pushwoosh. Check the full list of the Events offered by Adapty here.

3107.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Adapty will send subscription events to Pushwoosh using a server-to-server integration, allowing you to view all subscription events in your Pushwoosh Dashboard.

:::note Custom tags

With Adapty you can also use your custom tags for Pushwoosh integration. You can refer to the list of tags provided below to determine which tag is best suited for your needs. :::

| Tag | Type | Value | |---|----|-----| | `adapty_customer_user_id` | String | Contains the value of the unique identifier of the user, which can be found on the Pushwoosh side. | | `adapty_profile_id` | String | Contains the value of the unique identifier Adapty User Profile ID of the user, which can be found in your Adapty dashboard. | | `environment` | String |

Indicates whether the user is operating in a sandbox or production environment.

Values are either `Sandbox` or `Production`

| | `store` | String |

Contains the name of the Store that used to make the purchase.

Possible values:

`app_store` or `play_store`.

| | `vendor_product_id` | String |

Contains the value of Product ID in the Apple/Google store.

e.g., org.locals.12345

| | `subscription_expires_at` | String |

Contains the expiration date of the latest subscription.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `last_event_type` | String | Indicates the type of the last received event from the list of the standard Adapty events that you have enabled for the integration. | | `purchase_date` | String |

Contains the date of the last transaction (original purchase or renewal).

Value format is:
```

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `original_purchase_date` | String |

Contains the date of the first purchase according to the transaction.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | `active_subscription` | String | The value will be set to `true` on any purchase/renewal event, or `false` if the subscription is expired. | | `period_type` | String |

Indicates the latest period type for the purchase or renewal.

Possible values are

`trial` for a trial period or `normal` for the rest.

|

All float values will be rounded to int. Strings stay the same.

In addition to the pre-defined list of tags available, it is possible to send custom attributes using tags. This allows for more flexibility in the type of data that can be included with the tag and can be useful for tracking specific information related to a product or service. All custom user attributes are sent automatically to Pushwoosh if the user marks the ** Send user custom attributes** checkbox from the integration page

SDK configuration

To link Adapty with Pushwoosh, you need to send us the `HWID` value:

```swift let params = AdaptyProfileParameters.Builder() .with(pushwooshHWID: Pushwoosh.sharedInstance().getHWID()) .build()

Adapty.updateProfile(params: params) { error in // handle the error } </TabItem> <TabItem value="kotlin" label="Android (Kotlin)" default> kotlin val params = AdaptyProfileParameters.Builder() .withPushwooshHwid(Pushwoosh.getInstance().hwid) .build()

Adapty.updateProfile(params) { error -> if (error != null) { // handle the error } } </TabItem> <TabItem value="java" label="Java" default> java AdaptyProfileParameters params = new AdaptyProfileParameters.Builder() .withPushwooshHwid(Pushwoosh.getInstance().getHwid()) .build();

Adapty.updateProfile(params, error -> { if (error != null) { // handle the error } }); </TabItem> <TabItem value="Flutter" label="Flutter (Dart)" default> javascript import 'package:pushwoosh/pushwoosh.dart';

final builder = AdaptyProfileParametersBuilder() ..setPushwooshHWID( await Pushwoosh.getInstance.getHWID, ); try { await adapty.updateProfile(builder.build()); } on AdaptyError catch (adaptyError) { // handle error } catch (e) {} </TabItem> <TabItem value="Unity" label="Unity (C#)" default> csharp var builder = new Adapty.ProfileParameters.Builder(); builder.SetPushwooshHWID(Pushwoosh.Instance.HWID);

Adapty.UpdateProfile(builder.Build(), (error) => { // handle error }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript import { adapty } from 'react-native-adapty'; import Pushwoosh from 'pushwoosh-react-native-plugin';

// ... try { await adapty.updateProfile({ pushwooshHWID: hwid, }); } catch (error) { // handle `AdaptyError` } ```

---

title: "Webhook and ETL integrations" description: ""

## metadataTitle: ""

Check Adapty's step-by-step guides on how to integrate Adapty SDK with webhook and ETL options such as Amazon S3 and Google Cloud Storage.

With webhook integration, you can receive real-time notifications regarding user actions and events. These notifications can be customized and delivered to a preferred endpoint, enabling you to easily monitor and analyze your app's data. Use the following documentation to learn more about Adapty's webhook integration and how to implement it in your app,

- Webhook

Adapty offers a convenient feature that allows for automatic data deliveries of all transaction data related to your app. With this feature, you can effortlessly export transaction data to various cloud storage providers on a daily basis. The data is uploaded as a gzip compressed .csv file, which enables efficient storage and analysis of the information. Learn how to efficiently manage your data and streamline your data management by following our easy-to-use guides:

- Amazon S3
- Google Cloud Storage

---

title: "Amazon S3" description: ""

## metadataTitle: ""

Adapty's integration with Amazon S3 allows you to store event and paywall visit data securely in one central location. You will be able to save your subscription events to your Amazon S3 bucket as .csv files.
To set up this integration, you will need to follow a few simple steps in the AWS Console and Adapty dashboard.

:::note Schedule

Adapty sends your data every **24h** at 4:00 UTC.

Each file will contain data for the events created for the entire previous calendar day in UTC. For example, the data exported automatically at 4:00 UTC on March 8th will contain all the events created on March 7th from 00:00:00 to 23:59:59 in UTC. :::

## How to set up Amazon S3 integration

To start receiving data, you'll need the following credentials:

1. Access key ID
2. Secret access key
3. S3 bucket name
4. Folder name inside the S3 bucket

:::note Nested directories

You can specify nested directories in the Amazon S3 bucket name field, e.g. adapty-events/com.sample-app :::

To integrate Amazon S3 go to **Integrations -> Amazon S3**, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between Amazon S3 and Adapty profiles.

2023-03-24at_14.51.272x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

In the Adapty Dashboard, the following fields are needed to set up the connection:

| Field | Description | | :---------- | :---------- | | **Access Key ID** | A unique identifier that is used to authenticate a user or application's access to an AWS service. Find this ID in the downloaded csv file . | | **Secret Access Key** | A private key that is used in conjunction with the Access Key ID to authenticate a user or application's access to an AWS service. Find this Key in the downloaded csv file . | | **S3 Bucket Name \*** | ***A globally unique name that identifies a specific S3 bucket within the AWS cloud. S3 buckets are a simple storage service that allows users to store and retrieve data objects, such as files and images, in the cloud.*** | | ***Folder Inside the Bucker** | The name of the folder that you want to have inside the selected S3 bucket. Please note that S3 simulates folders using object key prefixes, which are essentially folder names. |

## How to create Amazon S3 credentials

This guide will help you create the necessary credentials in your AWS Console.
```

1. Create Access Policy

First, navigate to the IAM Policy Dashboard in your AWS Console and select the option to **Create Policy**.

2023-03-21at_10.52.002x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

In the Policy editor, paste the following JSON and change `adapty-s3-integration-test` to your bucket name:

```
json title="Json" { "Version": "2012-10-17", "Statement": [ { "Sid": "AllowListObjectsInBucket", "Effect": "Allow", "Action": "s3:ListBucket", "Resource": "arn:aws:s3:::adapty-s3-integration-test" }, { "Sid": "AllowAllObjectActions", "Effect": "Allow", "Action": "s3:*Object", "Resource": [ "arn:aws:s3:::adapty-s3-integration-test/*", "arn:aws:s3:::adapty-s3-integration-test" ] }, { "Sid": "AllowBucketLocation", "Effect": "Allow", "Action": "s3:GetBucketLocation", "Resource": "arn:aws:s3:::adapty-s3-integration-test" } ] }
```


2023-03-21at_10.56.212x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

After completing the policy configuration, you may choose to add tags (optional) and then click **Next** to proceed to the final step. In this step, you will name your policy and simply click on the **Create policy** button to finalize the creation process.

2023-03-21at_11.03.372x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Create IAM user

To enable Adapty to upload raw data reports to your bucket, you will need to provide them with the Access Key ID and Secret Access Key for a user who has write access to the specific bucket.

To proceed with that, navigate to the IAM Console and select the Users section. From there, click on the **Add users** button.

2023-03-21at_11.12.392x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Give the user a name, choose **Access key â€“ Programmatic access**, and proceed to permissions.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

For the next step, please select the **Add user to group** option and then click the **Create group** button.

2023-03-21at_11.24.592x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Next, you need to assign a name to your User Group and select the policy that was previously created by you. Once you have selected the policy, click on the **Create group** button to complete the process.

2023-03-21at_11.28.052x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

After successfully creating the group, please **select it** and proceed to the next step.

2023-03-21at_11.36.192x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Since this is the final step for this section, you may proceed by simply clicking on the **Create User** button.

2023-03-21at_11.40.462x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Lastly, you can either **download the credentials in .csv** format or alternatively, copy and paste the credentials directly from the dashboard.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Manual data export

In addition to the automatic event data export to Amazon S3, Adapty also provides a manual file export functionality. With this feature, you can select a specific time interval for the event data and export it to your S3 bucket manually. This allows you to have greater control over the data you export and when you export it.

The specified date range will be used to export the events created from Date A 00:00:00 UTC up to Date B 23:59:59 UTC.

2023-03-21at_12.35.252x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Table structure

In AWS S3 integration, Adapty provides a table to store historical data for transaction events and paywall visits. The table contains information about the user profile, revenue and proceeds, and the origin store, among other data points. Essentially, these tables log all transactions generated by an app for a given time period.

:::warning Note that this structure may grow over time â€” with new data being introduced by us or by the 3rd parties we work with. Make sure that your code that processes it is robust enough and relies on the specific fields, but not on the structure as a whole. :::

Here is the table structure for the events:

| Column | Description | |------|----------| |**profile*id*** | Adapty user ID. | | **eventtype** | Lower cased event name. Refer to the Events section to learn event types. | | **event*datetime*** | ISO 8601 date. | | **transactionid** | A unique identifier for a transaction such as a purchase or renewal. | | **original*transaction*id** | The transaction identifier of the original purchase. | | **subscription*expires*at** | The Expiration date of subscription. Usually in the future. | | **environment** | Could be Sandbox or Production. | | **revenue*usd*** | Revenue in USD. Can be empty. | | **proceedsusd** | Proceeds in USD. Can be empty. | | **net*revenue*usd** | Net revenue (income after taxes) in USD. Can be empty. | | **tax*amount*usd** | Amount of money deducted for taxes in USD. Can be empty. | | **revenue*local*** | Revenue in local currency. Can be empty. | | **proceedslocal** | Proceeds in local currency. Can be empty. | | **net*revenue*local** | Net revenue (income after taxes) in local currency. Can be empty. | | **tax*amount*local** | Amount of money deducted for taxes in local currency. Can be empty. | | **customer*user*id** | Developer user ID. For example, it can be your user UUID, email, or any other ID. Null if you didn't set it. | | **store** | Could be *app*store_ or *play*store. | | ***product*id** | Product ID in the Apple App Store, Google Play Store, or Stripe. | | **base*plan*id** | Base plan ID in the Google Play Store or price ID in Stripe. | | **developer*id*** | Developer (SDK) ID of the paywall where the transaction originated. | | **abtest*name*** | Name of the A/B test where the transaction originated. | | **abtest*revision*** | Revision of the A/B test where the transaction originated. | | **paywallname** | Name of the paywall where the transaction originated. | | **paywall*revision*** | Revision of the paywall where the transaction originated. | | **profilecounty** | Profile Country determined by Adapty, based on IP. | | **install*date*** | ISO 8601 date when the installation happened. | | **idfv** | identifierForVendor on iOS devices | | **idfa** | advertisingIdentifier on iOS devices | | **advertisingid** | The Advertising ID is a unique code assigned by the Android Operating System that advertisers might use to uniquely identify a user's device | | **ip*address*** | Device IP (can be IPv4 or IPv6, with IPv4 preferred when available). It is updated each time IP of the device changes. | | **cancellationreason** |

A reason why the user canceled a subscription.

Can be:

iOS & Android *voluntarily*cancelled, *_billing*error, *_refund*

iOS *price*increase, *_product*was*not*available, *_unknown*, *upgraded*

Android *new*subscription*replace*, *cancelled*by*developer*

| | **android*appsetid*** | An AppSetId - unique, per-device, per developer-account user-resettable ID for non-monetizing advertising use cases. | | **androidid** | On Android 8.0 (API level 26) and higher versions of the platform, a 64-bit number (expressed as a hexadecimal string), unique to each combination of app-signing key, user, and device. For more details, see Android developer documentation. | | **device** | The end-user-visible device model name. | | **currency** | The 3-letter currency code (ISO-4217) of the transaction. | | **store*country*** | Profile Country determined by Apple/Google store. | | **attributionsource** | Attribution source. | | **attribution*network*user*id*** | ID assigned to the user by attribution source. | | **attributionstatus** | Can be organic, non*organic or unknown.* | | ***attribution*channel** | Marketing channel name. | | **attribution*campaign*** | Marketing campaign name. | | **attributionad*group*** | Attribution ad group. | | **attributionad*set*** | Attribution ad set. | | **attributioncreative** | Attribution creative keyword. |

Here is the table structure for the paywall visits:

| Column | Description | | :---------- | :---------- | | **profile*id*** | Adapty user ID. | | **customeruser*id*** | Developer user ID. For example, it ca your user UUID, email, or any other ID. Null if you didn't set it. | | **profilecountry** | Profile Country determined by Apple/Google store. | | **install*date*** | ISO 8601 date when the installation happened. | | **store** | Could be _appstore_ or *play*store. | | ***paywall*showed*at*** | *The date when the paywall has been displayed to the customer.* | | ***developer*id** | Developer (SDK) ID of the paywall where the transaction originated. | | **abtest*name*** | Name of the A/B test where the transaction originated. | | **abtest*revision*** | Revision of the A/B test where the transaction originated. | | **paywall*name*** | Name of the paywall where the transaction originated. | | **paywallrevision** | Revision of the paywall where the transaction originated. |

Events and tags

Below the credentials, there are three groups of events you can export, send, and store in Amazon S3 from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-17at_14.49.282x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Google Cloud Storage" description: ""

metadataTitle: ""

Adapty's integration with Google Cloud Storage allows you to store event and paywall visit data securely in one central location. You will be able to save your subscription events to your Google Cloud Storage bucket as .csv files.
To set up this integration, you will need to follow a few simple steps in the Google Cloud Console and Adapty Dashboard.

:::note Schedule

Adapty sends your data to Google Cloud Storage every 24h at 4:00 UTC.

Each file will contain data for the events created for the entire previous calendar day in UTC. For example, the data exported automatically at 4:00 UTC on March 8th will contain all the events created on March 7th from 00:00:00 to 23:59:59 in UTC. :::

How to set up Google Cloud storage integration

To integrate Google Cloud Storage go to **Integrations -> Google Cloud Storage**, turn on a toggle from off to on, and fill out fields.

First of all set credentials to build a connection between Google Cloud Storage and Adapty profiles.

2023-03-17at_14.20.312x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

In the Adapty Dashboard, the following fields are needed to set up the connection:

| Field | Description | | :---------- | :---------- | | **Google Cloud Client ID** | A unique identifier assigned to your Google Cloud project when you create a new client in the Google Cloud Console. Fin this ID in the downloaded private JSON key file under the `client_id` field. | | **Google Cloud Project ID** | A user-assigned identifier for your Google Cloud project. Find this ID in the downloaded private JSON key file under the `project_id` field. | | **Google Cloud Service Account Private Key ID** | A unique identifier assigned to your private key when you create a new service account in the Google Cloud Console. Find this ID in the downloaded private JSON key file under the `private_key_id` field. | | **Google Cloud Bucket Name** | The name of the bucket in Google Cloud Storage where you want to store your data. It should be unique within the Google Cloud Storage environment and should not contain any spaces. | | ***Email** * | The email address associated with your service account in Google Cloud Console. It is used to grant access to resources in your project. | | ***Folder inside the bucket** * | The name of the folder inside the bucket where you want to store your data. It should be unique within the bucket and can be used to organize your data. This field is optional to fill. |

Create Google Cloud Storage credentials

This guide will help you create the necessary credentials in your Google Cloud Platform Console.

In order for Adapty to upload raw data reports to your designated bucket, the service account's key is required, as well as write access to the corresponding bucket. By providing the service account's key and granting write access to the bucket, you allow Adapty to securely and efficiently transfer the raw data reports from its platform to your storage environment.

:::warning Please note that we only support Service Account HMAC key authorization, means it's essential to ensure that your Service Account HMAC key has the "Storage Object Viewer", "Storage Legacy Bucket Writer" and "Storage Object Creator" roles added to it to enable proper access to Google Cloud Storage. :::

1. For the first step, you need to go to the IAM section of your Google Cloud account and choose the relevant project or create a new one.

2023-03-17at_15.22.142x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Next, create a new service account for the Adapty by clicking on the "+ CREATE SERVICE ACCOUNT" button.

2023-03-17at_15.40.062x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. Fill out the fields in the first step, as access will be granted at a later stage. In order to read more details about this page read the documentation here.

2023-03-17at_15.48.552x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. To create and download a private JSON key, navigate to the KEYS section and click on the "ADD KEY" button.

2023-03-17at_15.58.092x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

5. In the DETAILS section, locate the Email value linked to the recently created service account and make a copy of it. This information will be necessary for the upcoming steps to authorize the account and allow it to write to the bucket.

2023-03-17at_16.03.162x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. To proceed go to the Google Cloud Storage's Buckets page and either select an existing bucket or create a new one to store the Event or Visuts Data reports from Adapty. Then navigate to the PERMISSIONS section and select the option to GRANT ACCESS.

2023-03-17at_16.14.232x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

7. In the PERMISSIONS section, input the Email of the service account obtained in the fifth step mentioned earlier, then choose the Storage Object Creator role. Finally, click on SAVE to apply the changes.

2023-03-17at_16.17.312x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

```
Remember to keep the name of the bucket for future reference.
```

8. After passing these steps have successfully completed the necessary setup steps in the Google Cloud Console! The final step involves entering the bucket's name, accessing the JSON file containing the downloaded private key, and extracting the required field values for use in Adapty.

2023-03-17at_16.23.332x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Manual data export

In addition to the automatic event data export to Google Cloud Storage, Adapty also provides a manual file export functionality. With this feature, you can select a specific time interval for the event data and export it to your GCS bucket manually. This allows you to have greater control over the data you export and when you export it.

The specified date range will be used to export the events created from Date A 00:00:00 UTC up to Date B 23:59:59 UTC.

2023-03-17at_17.39.452x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Table structure

In Google Cloud Storage integration, Adapty provides table to store historical data for transaction events and paywall visits. The table contains information about the user profile, revenue and proceeds, and the origin store, among other data points. Essentially, these tables log all transactions generated by an app for a given time period.

:::warning Note that this structure may grow over time â€” with new data being introduced by us or by the 3rd parties we work with. Make sure that your code that processes it is robust enough and relies on the specific fields, but not on the structure as a whole. :::

Here is the table structure for the events:

| Column | Description | |------|----------| |**profile*id*** | Adapty user ID. | | **eventtype** | Lowercased event name. Refer to the Events section to learn event types. | | **event*datetime*** | ISO 8601 date. | | **transactionid** | A unique identifier for a transaction such as a purchase or renewal. | | **original*transaction*id** | The transaction identifier of the original purchase. | | **subscription*expires*at** | The Expiration date of subscription. Usually in the future. | | **environment** | Could be Sandbox or Production. | | **revenue*usd*** | Revenue in USD. Can be empty. | | **proceedsusd** | Proceeds in USD. Can be empty. | | **net*revenue*usd** | Net revenue (income after taxes) in USD. Can be empty. | | **tax*amount*usd** | Amount of money deducted for taxes in USD. Can be empty. | | **revenue*local*** | Revenue in local currency. Can be empty. | | **proceedslocal** | Proceeds in local currency. Can be empty. | | **net*revenue*local** | Net revenue (income after taxes) in local currency. Can be empty. | | **tax*amount*local** | Amount of money deducted for taxes in local currency. Can be empty. | | **customer*user*id** | Developer user ID. For example, it can be your user UUID, email, or any other ID. Null if you didn't set it. | | **store** | Could be *app*store_ or *play*store. | | ***product***id | Product ID in the Apple App Store, Google Play Store, or Stripe. | | **base*plan*id** | Base plan ID in the Google Play Store or price ID in Stripe. | | **developer*id*** | Developer (SDK) ID of the paywall where the transaction originated. | | **abtest*name*** | Name of the A/B test where the transaction originated. | | **abtest*revision*** | Revision of the A/B test where the transaction originated. | | **paywallname** | Name of the paywall where the transaction originated. | | **paywall*revision*** | Revision of the paywall where the transaction originated. | | **profilecountry** | Profile Country determined by Adapty, based on IP. | | **install*date*** | ISO 8601 date when the installation happened. | | **idfv** | identifierForVendor on iOS devices | | **idfa** | advertisingIdentifier on iOS devices | | **advertisingid** | The Advertising ID is a unique code assigned by the Android Operating System that advertisers might use to uniquely identify a user's device | | **ip*address*** | Device IP (can be IPv4 or IPv6, with IPv4 preferred when available). It is updated each time IP of the device changes | | **cancellationreason** |

A reason why the user canceled a subscription.

Can be:

iOS & Android*voluntarily*cancelled*, _billing*error*, _refund*

iOS *price*increase*, _product*was*not*available*, _unknown*, *upgraded*

Android *new*subscription*replace*, *cancelled*by*developer*

| | **android*appsetid*** | An AppSetId - unique, per-device, per developer-account user-resettable ID for non-monetizing advertising use cases. | | **androidid** | On Android 8.0 (API level 26) and higher versions of the platform,

a 64-bit number (expressed as a hexadecimal string), unique to each combination of app-signing key, user, and device. For more details, see Android developer documentation. | | **device** | The end-user-visible device model name. | | **currency** | The 3-letter currency code (ISO-4217) of the transaction. | | **store*country*** | Profile Country determined by Apple/Google store. | | **attributionsource** | Attribution source. | | **attribution*network*user*id*** | ID assigned to the user by attribution source. | | **attributionstatus** | Can be organic, non*organic or unknown.* | | ***attribution***channel | Marketing channel name. | | **attribution*campaign*** | Marketing campaign name. | | **attributionad*group*** | Attribution ad group. | | **attributionad*set*** | Attribution ad set. | | **attributioncreative** | Attribution creative keyword. |

Here is the table structure for the paywall visits:

| Column | Description | | :---------- | :---------- | | **profile*id*** | Adapty user ID. | | **customeruser*id*** | Developer user ID. For example, it ca your user UUID, email, or any other ID. Null if you didn't set it. | | **profilecountry** | Profile Country determined by Apple/Google store. | | **install*date*** | ISO 8601 date when the installation happened. | | **store** | Could be _appstore_ or _play_store. | | ***paywall***showed*at* | *The date when the paywall has been displayed to the customer.* | | ***developer***id | Developer (SDK) ID of the paywall where the transaction originated. | | **ab*test*name** | Name of the A/B test where the transaction originated. | | **ab*test*revision** | Revision of the A/B test where the transaction originated. | | **paywall*name*** | Name of the paywall where the transaction originated. | | **paywallrevision** | Revision of the paywall where the transaction originated. |

Events and tags

Below the credentials, there are three groups of events you can export, send, and store in Google Cloud Storage from Adapty. Simply turn on the ones you need. Check the full list of the events offered by Adapty here.

2023-08-17at_14.49.282x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

---

title: "Webhook integration" description: "Learn how to use webhooks to get instant notifications about subscription events like new trials and billing issues, enabling seamless automation and enhanced backend management for your app."

metadataTitle: "Understanding Webhooks: Real-Time Notifications for Subscription Events"

A webhook is a convenient and fast way to get notifications about events. For example, if a new trial is started, a subscription is renewed, or a billing issue happens. The webhook mechanism works through a callback function. You set up a URL (called Endpoint URL) to which an HTTP request is sent when an event occurs.

./ width: '700px', / image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

Unlike API requests, which require repeatedly requesting information from the server, a webhook is configured only once. When the target event occurs in your app, it immediately sends data via the HTTP protocol.

With webhooks integrated, you can:

- Keep track of subscriptions and purchases in your backend system.
- Automate processes and workflows based on subscription lifecycles.
- Engage with subscribers by reminding them of app benefits, addressing unsubscribe decisions, and handling billing issues.
- Conduct a detailed analysis of the user behavior.

Integration characteristics

| Integration characteristic | Description | | :---------- | :---------- | | Schedule | Real-time updates | | Data direction | One-way data transmission: from Adapty to your server | Adapty integration flow | Events are sent by the Adapty server once they are received |

Webhook event structure

Adapty will send you those events you've chosen in the **Events names** section of the **Integrations -> Webhooks** page.

Each event except for the access*level*update is wrapped into the structure below. Empty fields are not sent. For example, the line `"ad_group": null` will not be included into the package.

```
json title="Json" { "profile_id": "772204ce-ebf6-4ed9-82b0-d8688ab62b01", "customer_user_id": "john.doe", "idfv": "00000000-0000-0000-0000-000000000000", "idfa": "00000000-0000-0000-0000-000000000000", "advertising_id": "00000000-0000-0000-0000-000000000000", "profile_install_datetime": "2020-02-18T18:40:22.000000+0000", "user_agent": "Mozilla/5.0 (iPhone; CPU iPhone OS 15_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148", "email": "john.doe@company.com", "event_type": "non_subscription_purchase", "event_datetime": "2023-02-18T18:40:22.000000+0000", "event_properties": <event-specific properties>, "event_api_version": 1, "attributions": {"attribution_source1": <attribution_data>, "attribution_source2": <attribution_data>, ...}, "user_attributes": {"attribute_name1": "attribute_value1", "attribute_name2": "attribute_value2", ...} "integration_ids": {"firebase_app_instance_id": "val1", "branch_id": "val2", "one_signal_player_id": "val3", ... } }
```

Where

| Property | Type | Description | | :---------- | :---------- | :---------- | | **profile*id*** | String | The Ð¡ustomer user ID of the profile in Adapty. | | **customeruser*id*** | String | User ID you use in your app to identify the user. For example, it can be your user UUID, email, or any other ID. Null if you didn't set it. | | **idfv** | String | The identifier for vendors (IDFV) is a unique code assigned to all apps developed by a single developer, which in this case refers to your apps | | **idfa** | String | The identifier for advertisers (IDFA) is a random device identifier assigned by Apple to a user's device. | | **advertisingid** | String | The Advertising ID is a unique code assigned by the Android Operating System that advertisers might use to uniquely identify a user's device. | | **profile*install*datetime** | ISO 8601 date & time | Installation date and time in format IOS 8601: starting with the year, followed by the month, the day, the hour, the minutes, seconds, and milliseconds. For example, 2020-07-10T15:00:00.000000+0000, represents the 10th of July 2020 at 3 p.m. | | **user*agent*** | String | User-agent used by the browser on the device. | | **email** | String | E-mail of your user. | | **eventtype** | String | Event name as set up in the in the **Events names** section of the **Integrations -> Webhooks** page in lowercase. | | **event*datetime*** | ISO 8601 date & time | Event date and time in format IOS 8601 : starting with the year, followed by the month, the day, the hour, the minutes, seconds, and milliseconds. For example, 2020-07-10T15:00:00.000000+0000, represents the 10th of July 2020 at 3 p.m. | | **eventproperties** | JSON | JSON of event properties. | | **event*api*version** | Integer | Adapty API version. The current value is `1`. | | **attributions** | JSON | JSON of attribution data. | | **user*attributes*** | JSON | JSON of custom user attributes. | | **integrationids** | JSON | JSON of user integration identifiers. If a user doesn't have any identifier or integrations are disabled, then a null is sent. |

:::warning Note that this structure may grow over time â€” with new data being introduced by us or by the 3rd parties we work with. Make sure that your code that processes it is robust enough and relies on the specific fields, but not on the structure as a whole. :::

Webhook integration enables the control of sending attribution and user attributes.

- Enable the **Send Attribution** option in the **Integrations -> Webhooks** page to send the information about the source of app installs from data providers.
- Enable the **Send User Attributes** option to send custom user attributes set up from the Adapty SDK, such as user preferences and app usage data.

Event Access level updated

Adapty has a special event `access_level_updated`. It is sent only to webhook integration every time the access level is updated/set for a specific customer. Use this event to update a customer's subscription in your database/system. No matter what was the source of access level changes, you will always receive a dedicated event for that, therefore it's more precise and has more details than `subscription_renewed`, `trial_started`, `entered_grace_period`, etc.

2022-05-03at_14.22.56.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* image width */ *display: 'block',* for alignment */ *margin: '0 auto'* center alignment */ }} />

| Property | Type | Description | |--------|----|----------| **store** | str | Could be app*store, play*store, or stripe. | | **currency** | str | The 3-letter currency code (ISO-4217) of the transaction. | | **is*active*** | bool | Boolean indicating whether paid access level is active for the profile. | | **isrefund** | bool | Boolean indicating whether transaction is refunded. | | **expires*at*** | ISO 8601 date | Date and time when paid access will expire. | | **starts at** | ISO 8601 date | Date and time when paid access level starts for the user. | | **profileid** | str | Adapty user ID. | | **renewed*at*** | ISO 8601 date | Date and time when paid access will be renewed. | | **willrenew** | bool | Boolean indicating whether paid access level will be renewed. | | **environment** | str |

Indicates whether the user is operating in a sandbox or production environment.

Values are either Sandbox or Production.

| | **is*lifetime*** | bool | Boolean indicating whether paid access level is lifetime. | | **activatedat** | ISO 8601 date | Date and time when paid access was activated. | | **purchase*date*** | ISO 8601 date |

Contains the date of the last transaction (original purchase or renewal).

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | **storecountry** | str | The country sent to Adapty by the store. | | **event*datetime*** | ISO 8601 date | The date and time of the event. | | **transactionid** | str | A unique identifier for a transaction such as a purchase or renewal. | | **access*level*id** | str | Paid access level ID | | **profile*country*** | str | Profile Country determined by Apple/Google store. | | **profileevent*id*** | str | Unique event ID that can be used for deduplication. | | **vendorproduct*id*** | str |

Contains the value of Product Id in Apple/Google store.

e.g., org.locals.12345

| | **is*ingrace*period** | bool | Boolean indicating whether profile is in grace period. | | **original*purchase*date** | ISO 8601 date | The date and time of the original purchase. | | **original*transaction*id** | str | The transaction identifier of the original purchase. | | **subscription*expires*at** | ISO 8601 date |

Contains the expiration date of the latest subscription.

Value format is:

year-month dayThour:minute:second

e.g., 2023-02-10T17:22:03.000000+0000

| | **profile*total*revenue*usd*** | float | Total revenue for the profile, refunds included. | | **billingissue*detected*at** | ISO 8601 date | Date and time of billing issue. | | **cancellation*reason*** | str |

A reason why the user canceled a subscription.

Can be

iOS & Android:

- voluntarilycancelled

- billing*error*

- refund

iOS:

- priceincrease

- product*was*not*available*

- unknown

Android:

- newsubscription*replace*

- cancelledby*developer*

| | **activeintroductory*offer*type** | str | Type of the active introductory offer. | | **active*promotional*offer*type*** | str | Type of the active promotional offer. | | **activepromotional*offer*id** | str | ID of the active promotional offer as indicated in the Product section of the Adapty Dashboard |

Adapty doesn't send `access_level_updated` upon subscription expiration - please, refer to **expires_at** value to end the subscriptions on your side.

Please note that some properties can only be set using the grant Server-Side API method. You can find more information about this method in the Adapty Server-Side API documentation.

For detailed descriptions of the mentioned properties, you can refer to the API objects documentation.

Attribution data

If you've chosen to send attribution data, the following data will be sent with the event:

| Field name | Field type | Description | | :---------- | :---------- | :---------- | | **network*user*id** | str | ID assigned to the user by the attribution source. | | **status** | str | Can be `organic`, `non_organic` or `unknown`. | | **created*at*** | ISO 8601 date | Date and time of attribution record creation. | | **channel** | str | Marketing channel name. | | **campaign** | str | Marketing campaign name. | | **adgroup** | str | Attribution ad group. | | **ad_set** | str | Attribution ad set. | | **creative** | str | Attribution creative keyword. |

---

title: "Set up webhook integration" description: "Discover the step-by-step process to set up webhook integration in Adapty, enabling your server to receive real-time notifications for subscription events efficiently."

metadataTitle: "Set Up Webhook Integration for Real-Time Subscription Notifications"

Adapty webhook integration consists of the following steps:

1. You set up your endpoint to have the **Content-Type** header as `application/json`.
2. You configure the integration in the Adapty Dashboard. Within this step, you can optionally map Adapty events with your event names.
3. Adapty sends a verification request to your server.
4. Your server sends a verification response to Adapty
5. Test your webhook integration.

Configure webhook integration in the Adapty Dashboard

Within Adapty, you can configure separate flows for production events and test events received from the Apple or Stripe sandbox environment or Google test account.

For production events, use the **Production endpoint URL** field specifying the URL to which the callbacks will be sent. Additionally, configure the **Authorization header value for production endpoint** field - the header for your server to authenticate Adapty events. Note that we'll use the value specified in the **Authorization header value for production endpoint** field as the `Authorization` header exactly as provided, without any changes or additions.

For test events, employ the **Sandbox endpoint URL** and **Authorization header value for sandbox endpoint** fields accordingly.

To set up the webhook integration:

1. Open **Integrations -> Webhook** in your Adapty Dashboard.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Turn on the toggle to initiate the integration.
3. Fill out the integration fields:

| Field | Description | |-----|----------| |**Production endpoint URL** | URL that is used by Adapty to send HTTP POST requests to this URL when events occur. | | **Authorization header value for production endpoint** |

The header that your server will use to authenticate requests from Adapty in production. Note that we'll use the value specified in this field as the `Authorization` header exactly as provided, without any changes or additions.

Although not mandatory, it's strongly recommended for enhanced security.

|

Additionally, for your testing needs in the staging environment, two other fields are available:

| Testing field | Description | |----------|----------| **Sandbox endpoint URL** | Adapty will use this URL to send HTTP POST requests when events occur in the staging environment. | | **Authorization header value for sandbox endpoint** |

The header that your server will use to authenticate requests from Adapty during testing in the sandbox environment. Note that we'll use the value specified in this field as the `Authorization` header exactly as provided, without any changes or additions.

Although not mandatory, it's strongly recommended for enhanced security.

| 4. Choose the events you want to receive and map their names. 5. Additional fields and options are not obligatory; use them as needed. 6. Remember to click the **Save** button to confirm the changes.

Please note that the moment you click the **Save** button, Adapty will send a verification request and will wait for your server verification response.

Choose events to send and map event names

Choose the events you want to receive in your server by enabling the toggle next to it. If your event names differ from those used in Adapty and you need to keep your names as is, you can set up the mapping by replacing the default Adapty event names with your own in the **Events names** section of the **Integrations -> Webhooks** page.

namesrenaming.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

The event name can be any string. You cannot leave the fields empty for enabled events. If you accidentally removed Adapty event name, you can always copy the name from the Events to send to 3d-party integrations topic.

Adapty's verification request

After you enable webhook integration in the Adapty Dashboard, Adapty will automatically send a `isMount` POST verification request to your endpoint.

```
json title="Json" { adapty_check: {{check_string}} }
```

:::note Be sure your endpoint supports **Content-Type**: `application/json` header :::

Your server's verification response

Your server must reply with a 200 or 201 HTTP status code and send the response outlined below with the identical `check_string`.

```
json title="Json" { adapty_check_response: {{check_string}} }
```

Once Adapty receives the verification response in the correct format, your Adapty webhook integration is fully configured.

From then on, Adapty will send the selected events to your specified URL as they happen. Ensure your server promptly confirms each Adapty event with a response status code in the 200-404 range.

Handle webhook events

Webhooks are typically delivered within 5 to 60 seconds of the event occurring. Cancellation events, however, may take up to 2 hours to be delivered after a user cancels their subscription.

If your server's response status code is outside the 200-404 range, Adapty will retry sending the event up to 9 times over 24 hours with exponential backoff. We suggest you set up your webhook to do only basic validation of the event body from Adapty before responding. If your server can't process the event and you don't want Adapty to retry, use a status code within the 200-404 range. Also, handle any time-consuming tasks asynchronously and respond to Adapty quickly. If Adapty doesn't receive a response within 10 seconds, it will consider the attempt a failure and will retry

---

title: "Test webhook integration" description: "Learn how to test your Adapty webhook integration, validate event delivery, and ensure correct setup for historical events, subscriptions, and more"

metadataTitle: "Testing Your Webhook Integration with Adapty"

After you set up your integration, it's time to test it. You can test both your sandbox and production integration. We recommend starting with the sandbox one and validating the maximum on it:

- The events are sent and successfully delivered
- You set up the options correctly for historical events, subscription price for the **Trial started** event, attribution, user attributes, and Google Play Store purchase token to be sent or not sent with an event
- You mapped event names correctly and your server can process them

How to test

Before you start testing an integration, make sure you have already:

1. Set up the webhook integration as described in the Set up webhook integration topic.
2. Set up the environment as described in the Test in-app purchases in Apple App Store and Test in-app purchases in Google Play Store topics. Make sure you built your test app in the sandbox environment rather than in the production one.

To test the integration:

Make a purchase/start a trial/make a refund that will raise an event you've chosen to send to the webhook. For example, to get the **Subscription started** event, purchase a new subscription.

Validation of the result

Successful sending events result

In case of successful integration, an event will appear in the **Last sent events** section of the integration and will have the **Success** status.

integrationsuccess.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

Unsuccessful sending events result

| Issue | Solution | |-----|--------| | The event did not appear | Your purchase did not occur and therefore the event was not created. Refer to theTroubleshooting test purchases topic for the solution. | | The event appeared and has the **Sending failed** status |

We determine the deliverability based on HTTP status and consider everything **outside the 200-399 range** to be a fail.

To find more on the issue, hover over the **Sending failed** status of your unsuccessful event as shown below, then consult the Integration Event Sending Failures table to find the solution.

|

sendingfailed.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

---

title: "Stripe integration" description: ""

metadataTitle: ""

Adapty supports tracking web payments and subscriptions made through Stripe. If you're already offering your product on the web or thinking about doing it, there are two scenarios where it can be helpful:

- automatically providing access to paid features for users who purchased on the web but later installed the app and logged in to their account
- having all the subscription analytics in a single Adapty Dashboard (including cohorts, predictions and the rest of our analytics toolbox)

Even though purchases on the web are becoming increasingly popular for apps, you must remember that it is against Apple's App Store terms to provide a different system than in-app purchases for digital goods. Make sure you don't promote your web subscriptions from inside your app. Otherwise, your app may get rejected or banned.

The steps below outline how to configure the integration with Stripe.

1. Connect Stripe to Adapty

This integration mainly relies on Adapty pulling subscription data from Stripe via the webhook. So it is essential to connect your Adapty account to your Stripe account by doing 2 things: providing API Keys and using Adapty's webhook URL in Stripe.

:::note The steps below are the same for Production and Sandbox (or Test mode in Stripe). You can do them simultaneously for both environments. :::

1. Go to Developers â†’ API Keys in Stripe:

2023-12-06at_17.29.122x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

2. Click the **Reveal live (test) key button** next to the **Secret key** title, then copy it and go to Adapty's **App Settings â†’ Stripe**. Paste the key here:

2023-12-07at_14.59.122x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

3. Next, copy the Webhook URL from the bottom of the same page in Adapty. Go to **Developers â†’ Webhooks** in Stripe and click the **Add endpoint** button:

2023-12-07at_17.31.392x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

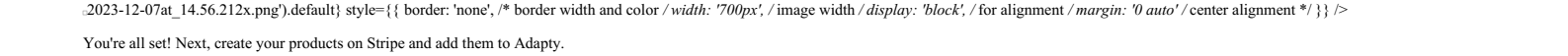
4. Paste the webhook URL from Adapty into the **Endpoint URL** field. Then choose the **Latest API version** in the webhook **Version** field. Then select the following events:

    - charge.refunded
    - customer.subscription.created
    - customer.subscription.deleted
    - customer.subscription.paused
    - customer.subscription.resumed
    - customer.subscription.updated
    - invoice.created
    - invoice.updated
    - payment_intent.succeeded

2023-12-07at_17.36.232x.png').default} style={{ border: 'none', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

5. Press "Add endpoint" and then press "Reveal" under the "Signing secret". This is the key that is used to decode the webhook data on the Adapty's side, copy it after revealing:

6. Finally, paste this key into Adapty's App Settings â†’ Stripe under "Stripe Webhook Secret":

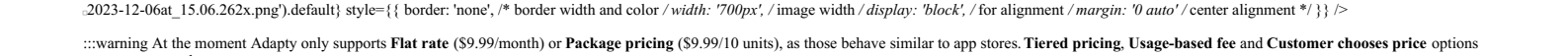


You're all set! Next, create your products on Stripe and add them to Adapty.

2. Create products on Stripe

:::note If you're configuring Sandbox, make sure to switch to Test mode in Stripe, before proceeding with this step. :::

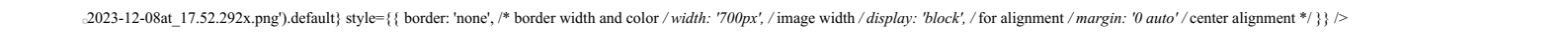
Go to Stripe's Product catalog and create the products you would like to sell as well as their pricing plans. Note that Stripe allows you to have multiple pricing plans per product, which is useful for tailoring your offering without the need to create additional products.



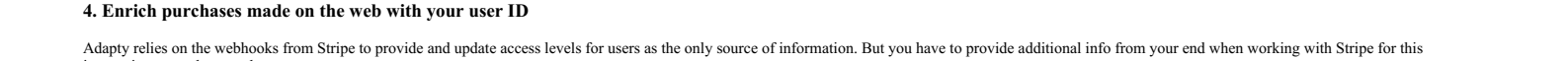
:::warning At the moment Adapty only supports **Flat rate** ($9.99/month) or **Package pricing** ($9.99/10 units), as those behave similar to app stores. **Tiered pricing**, **Usage-based fee** and **Customer chooses price** options are not supported :::

3. Add Stripe products to Adapty

We treat Stripe the same way as App Store and Google Play: it is just another store where you sell your digital products. So it is configured similarly: simply add Stripe products (namely their `product_id` and `price_id`) to the Products section of Adapty:



Product IDs in Stripe look like `prod_...` and price IDs look like `price_...`. They are pretty easy to find for each product in Stripe's Product Catalog, once you open any Product:



After you've added all the necessary products, the next step is to let Stripe know about which user is making the purchase, so it could get picked up by Adapty!

4. Enrich purchases made on the web with your user ID

Adapty relies on the webhooks from Stripe to provide and update access levels for users as the only source of information. But you have to provide additional info from your end when working with Stripe for this integration to work properly.

For access levels to be consistent across platforms (web or mobile), you have to make sure that there is a single user ID to rely on which Adapty can recognize from the webhooks as well. This could be user's email, phone number or any other ID from the authorization system you're utilizing.

Figure out which ID you would like to use to identify your users. Then, access the part of your code that's initiatilizing the payment through Stripe â€” and add this user ID to the `metadata` object of either Stripe Subscription (`sub_...`) or Checkout Session object (`ses_...`) as `customer_user_id` like so:

```
json title="Stripe Metadata contents" {'customer_user_id': "YOUR_USER_ID"}
```

This one simple addition is the only thing that you have to do in your code. After that, Adapty will parse all the webhooks it receives from Stripe, extract this `metadata` and correctly associate subscriptions with your customers.

:::warning User ID is required

Otherwise, we have no way to match this user and provide him the access level on the mobile.

If you don't supply `customer_user_id` to the `metadata`, you will have the option to make Adapty look for `customer_user_id` in other places: either `email` from Stripe's Customer object or `client_reference_id` from Stripe's Session.

Learn more about configuring profile creation behavior below :::

:::note Customer in Stripe is also required

If you are using Checkout Sessions, make sure you're creating a Stripe Customer by setting `customer_creation` to `always`. :::

5. Provide access to users on the mobile

To make sure your mobile users arriving from web can access the paid features, just call `Adapty.activate()` or `Adapty.identify()` with the same `customer_user_id` you've provided on the previous step (see Identifying users for more).

6. Test your integration

Make sure you've completed the steps above for Sandbox as well as for Production. Transactions that you make from Stripe's Test mode will be considered Sandbox in Adapty.

:::info That's it!

Your users can now complete purchases on the web and access paid features in your app. And you can also see all your subscription analytics in a single place. :::

Profile creation behavior

Adapty has to tie a purchase to a customer profile for it to be available on the mobile â€” so by default it creates profiles upon receiving webhooks from Stripe. You can choose what to use as customer user ID in Adapty:

1. ***Default and recommended:*** `customer_user_id` you supplied in metadata in step 4 above
2. `email` in Stripe's Customer object (see Stripe's docs)
3. `client_reference_id` in Stripe's Session object (see Stripe's docs)

You can configure which ID you would like to use in App Settings â†’ Stripe.

:::warning **Note:** if a particular transaction from Stripe does not contain the specified ID, we will not create a profile at all. This transaction will remain anonymous until it gets picked up by some profile (for example, if you use S2S validate afterwards and tell us about this transaction manually).

It will show up in Analytics but not in the sections that rely on counting profiles (LTV, Cohorts, Conversions, etc) and you won't be able to see it in Event feed. :::

You also have a fourth option to not create profiles at all but this is not recommended due to the above limitations in Analytics.

Current limitations

Upgrading, downgrading and proration

Subscription changes such as upgrading or downgrading can result in prorated charges. Adapty will not account for these charges in revenue calculations. It would be best to disable these options manually via the Stripe dashboard. You can also disable them by setting the `proration_behaviour` attribute value to `none` via the Stripe API.

Cancellations

Stripe has two subscription cancellation options:

1. Immediate cancellation: The subscription cancels immediately with or without any proration option
2. Cancellation at the end of the period: The subscription cancels at the end of the current billing period (similar to in-app subscriptions on the app stores).

Adapty supports both options, but the revenue calculation for immediate cancellation will disregard the proration option.

Billing Issues and Grace Period

When a customer encounters an issue with their payment, Adapty will generate a billing issue event and access will be revoked. We do not support Stripe's Grace Period just yet â€” this will be a part of future releases.

Refunds

Adapty tracks only full refunds. Proration or partial refunds are currently not supported.

---

title: "App settings" description: ""

metadataTitle: ""

You can navigate to the General tab of the App Settings page to manage your app's behavior, appearance, and revenue sharing. Here, you can customize your app's name and icon, manage your Adapty SDK and API keys, set your Small Business Program status, and choose the timezone for your app's analytics and charts.

1. App details

2023-04-21at_15.16.222x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Choose a unique name and icon that represent your app in the Adapty interface. Please note that the app name and icon will not affect the app's name and icon in the App Store or Google Play. Also, make sure to select an appropriate App Category that accurately reflects your app's purpose and content. This will help users discover your app and ensure it appears in the appropriate app store categories.

2. Member of Small Business Program and Reduced Service Fee

2023-04-19at_13.43.292x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

If you're a member of the Apple Small Business Program and/or Google's Reduced Service Fee program, you can let Adapty know by specifying the period that you are a member. Adapty will adjust the commission rate accordingly, so you can keep more of your revenue. Please note that this setting applies only to future transactions, and you need to update it if your Small Business Program status changes. You can learn more about the App Store Small Business Program and Google's Reduced Service Fee.

3. Reporting timezone

2023-04-19at_13.45.302x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Choose the timezone that corresponds to the location where you're based, or where your app's analytics and charts are most relevant. We recommend using the same timezone as your App Store Connect or Google Play Console account to ensure consistency. Please note that this timezone setting does not affect third-party integrations in the Adapty system, which use the UTC timezone.

You can access the timezone settings in the Reported timezone section of the General Tab on the App Settings page. You can also choose to set the same timezone for all the apps in your Adapty account by checking the corresponding box.

4. App Store price increase logic

To maintain accurate data and avoid discrepancies between Adapty analytics and App Store Connect results, it is important to select the appropriate option when adjusting configurations related to price increases in App Store Connect.
So you can choose the logic that will be applied to subscription price increases in Adapty:

2023-07-18at*19.28.18*22x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

- **Subscription price for existing users is preserved:** By selecting this option, the current price will be retained for your existing subscribers, even if you make changes to the price in the App Store Connect. This means that existing subscribers will continue to be billed at their original subscription price.
- **When the subscription price is changed in App Store Connect, it changes for existing subscribers:** If you choose this option, any price changes made in the App Store Connect will be applied to your existing subscribers as well. This means that existing subscribers will be charged the new price reflecting the updated pricing set in the App Store Connect.

:::warning It is important to consider that the selected option not only affects analytics in Adapty but also impacts integrations and overall transaction handling behavior. :::

Please ensure that you select the designated option that aligns with your desired approach to handling subscription prices for existing subscribers. This will help maintain accurate data and synchronization between Adapty analytics and the results obtained from the App Store Connect.

5. Sharing purchases between user accounts

This setting determines what happens when Adapty receives a purchase from a Customer User ID that is currently associated with another Customer User ID.

Sharing is enabled by default â€” meaning that anonymous and identified users can share the same access level provided by Adapty if the app store on their device is under the same Apple/Google ID. This can be helpful for example when your user re-installs the app and chooses to log in under a different email â€” in that case, they will still get access to their previous purchase.

Apple and Google require "sharing" in-app purchases between your users by default, because they rely on their Apple/Google IDs to tie the purchase to â€” and otherwise restoring would not work on some reinstalls.

However, you might want to disable sharing between different user accounts in case you rely on an internal login system and you would like to prevent two people from using the same purchase.

Here is what happens when this option is disabled:

- Access level is still shared between anonymous users (that is if a user never gets identified and assigned a customer user ID)
- When Adapty first sees a customer user ID connected to the original purchase (for example, when a user logs in or signs up), this purchase becomes "owned" by this customer user ID.
- After that, this purchase is only available to the original user. If another user (anonymous or identified) comes along with the same Apple/Google ID after a reinstall â€” Adapty will not provide access to them.
- You can "untie" the purchase only by deleting the owner's user profile. After deletion, this access level becomes available to the first user profile to claim it (anonymous or identified).

That way you can make sure there is only one user profile for every subscription.

:::warning Disabled sharing may result in some of your users not getting access on login

We advise you only consider disabling sharing if your users **are required to login** before they get a chance to make a purchase. Otherwise there could be cases where a user purchases a subscription, logs into an existing account and loses access once and for all. :::

Note: Disabling sharing will only affect the new users. Subscriptions that have already been shared between existing users will continue to be shared after you enable this setting.

6. SDK and API keys

Use a Public SDK key to integrate Adapty SDKs into your app, and a Secret Key to access Adapty's Server API. You can generate new keys or revoke existing ones as needed.

7. Test devices

Specify the devices to be used for testing to ensure they get instant updates for paywall or placement changes, bypassing any caching delays. For more information, see Testing devices.

8. Delete the app

If you no longer need an app, you can delete it from Adapty.

:::warning Please be aware that this action is irreversible, and you won't be able to restore the app or its data. :::

---

title: "Apple App Store credentials" description: ""

metadataTitle: ""

To configure the App Store credentials and ensure optimal functionality of the Adapty iOS SDK, navigate to the iOS SDK tab within the App Settings page of the Adapty Dashboard. Then, configure the following parameters:

2023-06-26at_13.27.042x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

| Field | Description | |-----|----------| |**Bundle ID** | Your app bundle ID | | **In-app purchase API (StoreKit 2)** | Keys to enable secure authentication and validation of in-app purchase transaction history requests. | | **App Store Server Notifications** | URL that is used to enable server2server notifications from the App Store to monitor and respond to users' subscription status changes | | **App Store Promotional Offers** | Subscription keys for creating Promotional offers in Adapty for specific products. | | **App Store Connect shared secret (LEGACY)** |

Legacy key for StoreKit 1 and Adapty SDK prior to v.2.9.0

A key for receipts validation and preventing fraud in your app.

|

---

title: "Apple In-App Purchase API (StoreKit 2)" description: ""

metadataTitle: ""

For Adapty to securely validate in-app purchases in your app, and authenticate and validate subscription requests with Apple, you need to upload an in-app purchase key and provide the Issuer ID and Key ID.

1. Generate the API key

:::note To generate API keys for the App Store Server API, you need to have either an Admin role or Account Holder role in App Store Connect. Read more about how to generate API Keys here. :::

1. Open **App Store Connect**. Proceed to **Users and Access â†' Integrations â†' In-App Purchase** section.
2. Then click the **Plus (+)** button next to the **Active** title. The **Generate in-App Purchase Key** window will open.

3. Enter the name of the key for your future reference (this name is not used in Adapty).
4. Click the **Generate** button. Once the **Generate in-App Purchase Key** window closes, you'll see the created key in the **Active** list.

5. After generating your API key, click the **Download In-App Purchase Key** button to get the key as a file.
   Keep this file safe to later upload it to Adapty. Please keep in mind that the generated file can be downloaded only once, so store it safely until you upload it to Adapty. The generated .p8 key from the **In-App Purchase** section can be used for both In-app purchase API and promotional offers.
6. Copy the values of **Issuer ID** and **Key ID** fields from this screen as well. Later you will use them in Adapty.

2. Add the generated API key to Adapty

After completing the necessary setup steps in App Store Connect, the next step is to upload the In-App Purchase Key to Adapty:

1. Open **App settings -> iOS SDK** in your Adapty dashboard.
2. Scroll down to the **In-App Purchase API (StoreKit 2)** section.
3. Fill out the **Issuer ID** and the **Key ID** **** with the corresponding field values copied from your App Store Connect. Please note that the fields will only be active after the app's *Bundle ID*** is provided, as the in-app purchase key is specific to each app bundle.

4. Upload the **In-App Purchase Key** file in .p8 format you've downloaded from App Store Connect to the***** *Private Key (.p8 file)******** field.

After completing these steps, you're all set!

If you are currently using StoreKit 1 notifications and you are now configuring StoreKit 2, it is recommended that you switch to V2 notifications on the App Store Connect side as well. You can read more about it in App Store server notifications.

---

title: "Apple App Store server notifications" description: ""

metadataTitle: ""

Apple offers server-to-server notifications, so you can instantly be notified about subscription events.

Adapty helps you with that. The only thing you need to do is to set the URL for App Store Server Notifications inside your App Store Connect to Adapty status URL.

:::note Subscription status URL is per app

This URL is specific to each of your apps. So if you have multiple apps, you need to set different URLs. :::

Sending App Store server notifications to Adapty

1. Copy URL for App Store Server Notifications in Adapty Dashboard **App Settings -> iOS SDK**

2. Sign in to your App Store Connect account, choose the app, and go to the **App Information** page in section **General**. Use the **URL from Adapty** for both **Production** and **Sandbox** notifications, and save the changes.

Adapty supports both **Version 1** and **Version 2** Notifications. You can choose whichever version best suits your needs. To learn more about the differences between the two versions, please refer to this link. You can also check out our tutorial to learn details about the version.

Please note that to use **Version 2** Notifications, it is necessary to configure the In-App Purchase API details as described in the documentation. This step is essential for ensuring that Version 2 Notifications work correctly.

Raw events forwarding

Sometimes, you might still want to receive raw S2S events from Apple. To continue receiving them while using Adapty, just add your endpoint to the **URL for forwarding raw Apple events** field, and we'll send raw events as-is from Apple.

---

title: "Apple App Store Promotional Offers" description: ""

metadataTitle: ""

Offers in the App Store are special deals or discounts provided by these platforms for in-app purchases. For example, you can offer a user upfront payment for 6 months with a 40% discount, and after that user will pay the regular subscription price every month.

For Adapty to process offers from the App Store, you need to do the following:

1. Create offers in the App Store Connect or create offers in the Google Play Console
2. Create offers in Adapty
3. Add these offers to a paywall in Adapty
4. (only for iOS apps) Upload a special In-App Purchase Key from App Store Connect to Adapty.

The instructions on how to upload a special In-App Purchase Key from App Store Connect to Adapty are below.

:::note - Please note that you can use the same key for both In-app purchase API and promotional offers â€" there's no need to create a new one if you have access to the .p8 file you've previously generated. - To generate API keys for the App Store Server API, you need to have either an Admin role or an Account Holder role in App Store Connect. You can also read about how to generate API Keys in the Apple Developer Documentation. :::

1. Open **App Store Connect**. Proceed to **Users and Access â†' Integrations â†' In-App Purchase** section.
2. Then click the **Plus (+)** button next to the **Active** title. The **Generate in-App Purchase Key** window will open.

3. Enter the name of the key for your future reference (this name is not used in Adapty).

4. Click the **Generate** button. Once the **Generate In-App Purchase Key** window closes, you'll see the created key in the **Active** list.

5. After generating your API key, click the **Download In-App Purchase Key** button to get the key as a file.
   Keep this file safe to later upload it to Adapty. Please keep in mind that the generated file can be downloaded only once, so store it safely until you upload it to Adapty. The generated .p8 key from the **In-App Purchase** section can be used for both In-app purchase API and promotional offers.

6. Copy the value of the **Key ID** field from this screen as well.

7. Open **App Settings** from Adapty top menu, then open the iOS SDK tab.

8. Enter the value of the copied **Key ID** field to the **Subscription key ID** field next to the **App Store promotional offers** title.

storepromotional*offers*in_adapty.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

9. Upload the downloaded In-App Purchase Key file to the **Subscription key (.p8 file)** area .

After configuring the App Store promotional offer, you can create Apple promotional offers for specific products in Adapty if you did not do that yet. For more information, please refer to our promotional offers documentation.

---

title: "Apple App Store shared secret (LEGACY)" description: ""

metadataTitle: ""

:::warning This is a legacy receipt verification method

You only need to configure this shared secret if you're on Apple's StoreKit <v2.0 and Adapty SDK <v.2.9.0 â€" as it is currently deprecated by Apple.

If you're just starting out with Adapty, configure Apple In-App Purchase API instead. :::

Adapty uses this key for receipt verification. This key is app-specific, make sure to generate it for each of your apps. To do this please follow the steps below.

:::note You can also generate one Primary Shared Secret, and use one key for all your apps. To generate it, go to Users and Access > Shared Secret page and click **Generate** there. :::

1. Generate a shared secret for your app

Select your app on the App Store Connect apps page. Go to **App Information** in section **General**. On the page, you can see the App-Specific Shared Secret description with **Manage** link below, click it, and you'll be able to see or create a new shared secret.

2023-08-25at*12.14.41*22x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

Generate a Shared Secret, copy it, and don't forget to paste it in Adapty Dashboard.

2023-08-25at_12.15.562x.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

2. Add the generated shared secret to Adapty

Select App settings -> iOS SDK in Adapty. Scroll down to App Store Connect shared secret section, and enter your shared secret.

2022-12-29at_07.53.55.png').default} style={{ border: '1px solid #727272', /* border width and color */ *width: '700px',* /* image width */ *display: 'block',* /* for alignment */ *margin: '0 auto'* /* center alignment */ }} />

---

title: "Google Play Store credentials" description: ""

metadataTitle: ""

For Adapty Android SDK to work, you need to configure several parameters.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

| Field | Description | | :---------- | :---------- | | **Package nar
Package name is the unique identifier of your app in the Google Play Store. This is required for the basic functionality of Adapty, such as subscription processing. | | **Service account key file** | Keys to enable secure authentication and validation of purchases. | | **Google Play RTDN topic name** | URL that is used to enable server2server notifications from the Play Store to monitor and respond to users' subscription status changes. |

---

title: "Google Play Store credentials" description: ""

metadataTitle: ""

Google Play Store credentials are used for server-side purchase verification, without them Adapty can't validate and process transactions. Obtaining the credentials is a bit tricky but this guide will help you.

1. Enable APIs in the Google Cloud Console

1. Select the Google Cloud Project you want to use or create a new one. Please make sure to use the same Google Cloud Project during all the steps of this guide.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

2. Go to the Google Play Android Developer API page and click **Enable**.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

3. Go to the Google Play Developer Reporting API page and click **Enable**.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

2. Create a Service account in the Google Cloud Console

1. Go to Google Cloud Console -> IAM & Admin -> Service accounts. Please make sure to use the same Google Cloud Project you used in the previous steps. Click **Create Service Account**.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

2. Enter its name and copy its **ID** (email address) for future use. Click **Create and Continue**.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

3. In the next step you should add two **Roles**:

- Pub/Sub Admin (to enable Real-time Developer Notifications)
- Monitoring Viewer (to allow monitoring of the notification queue).

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

4. Don't enter anything in this step and click **Done**.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

5. Find the newly created account in the list and in the actions click **Manage keys**. Create a new JSON key and save it locally on your computer. This is the key you will upload to the Adapty Dashboard.

/ width: '700px', / image width */ *display: 'block',* / for alignment */ *margin: '0 auto'* / center alignment */ }} />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

3. Grant permissions in the Google Play Console

1. Go to the **Users and permissions** page in the Google Play Console and click **Invite new users**.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

2. Enter the email of the service account you've created in section 2 of this guide.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

3. Grant **View app information and download bulk reports (read-only)**, **View financial data, orders, and cancellation survey responses**, **Manage orders and subscriptions** and **Manage store presence** permissions and create the user by clicking **Invite user**. You can grant permissions to the whole account or to one/several applications.

2021-08-31at_23.26.08.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

4. Upload Google credentials to Adapty Dashboard

1. Go to Adapty -> App Settings -> Android SDK. Upload the **Service Account Key File** (JSON file) to the Adapty Dashboard. Make sure to set a **Package Name** too. If you've done everything correctly, Adapty will generate an RTDN topic for you. Copy it and set up Real-time Developer Notifications.

2023-08-30at_16.02.482x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

:::note It takes at least 24 hours for changes to take effect but there's a hack. In Google Play Console, open any application and in the **Monetize** section go to **Products** -> **Subscriptions/In-app products**. Change the description of any product and save the changes. Everything should be working now, you can revert in-app changes. If not, make sure **Google Play Android Developer API** is enabled in Google Cloud Project. :::

---

title: "Google Real-time developer notifications (RTDN)" description: ""

metadataTitle: ""

Real-time developer notifications RTDN allow receiving updates the moment they occur in the Play Store, tracking refunds, and more. You must set them up for data accuracy.

:::warning Make sure that Cloud Pub/Sub API is enabled and your

Service Account Key File has **Pub/Sub Admin** permissions. :::

2023-08-30at_15.22.462x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

After uploading a Service Account Key File to Adapty, we will automatically create all needed resources in your Google Cloud Project, and the topic name will appear in the **Google Play RTDN topic name** field on Android SDK settings page.

2023-08-25at_12.46.002x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Copy and paste its value into the **Topic name** field found at App Dashboard -> **Monetization setup** page of Google Play Console, click **Send test notification** to make sure everything works, and save the changes.

2023-08-25at_12.47.442x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Go back to Android SDK settings page and refresh it. The status next to Google Play RTDN topic name should be changed to **Active**. If it hasn't changed, make sure you clicked **Send test notification** and indicated the right topic name.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Raw events forwarding

Sometimes, you might still want to receive raw S2S events from Google. To continue receiving them while using Adapty, just add your endpoint to the **URL for forwarding raw Google events** field, and we'll send raw events as-is from Google.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

---

title: "Account details" description: ""

metadataTitle: ""

Your account page can be accessed through the link in the top right corner after you've logged into Adapty. Or just by this link.

There are several important things you can set on your account page so let's go through them.

2022-12-21at_20.47.10.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

General settings

Here you fill in your name and the name of your company. It's useful to have this info when you contact support so they can work with the correctly filled profile.

2022-12-22at_08.24.40.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ }} />

Your main credentials are also on this part of the page.
So you may change your password here or check what e-mail your account is registered for.

Date and time format

Being a part of general settings date and time format deserves to be highlighted separately because it can make your life with Adapty easier.
By selecting the option you can choose American or European format to have date and time in a convenient way.
For example, if you prefer January 31, 2022 date format and it's common for you to use AM and PM in time your choice is the American format.
And vice versa if it's common for you to use 31 January, 2022 date format and see the time as 16:00 then your option is European format.

Members

You can manage your team members in your account settings. Please read more about it.

Billing info

You can edit your payment method and upgrade your plan on the account page.

---

title: "Account" description: ""

metadataTitle: ""

Check our step-by-step guides to learn more about your account, subscription, and members management in the Adapty Dashboard:

- Account details
- Members

---

title: "Account details" description: ""

metadataTitle: ""

Your account page can be accessed through the link in the top right corner after you've logged into Adapty. Or just by this link.

There are several important things you can set on your account page so let's go through them.

2022-12-21at_20.47.10.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

General settings

Here you fill in your name and the name of your company. It's useful to have this info when you contact support so they can work with the correctly filled profile.

2022-12-22at_08.24.40.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Your main credentials are also on this part of the page.
So you may change your password here or check what e-mail your account is registered for.

Date and time format

Being a part of general settings date and time format deserves to be highlighted separately because it can make your life with Adapty easier.
By selecting the option you can choose American or European format to have date and time in a convenient way.
For example, if you prefer January 31, 2022 date format and it's common for you to use AM and PM in time your choice is the American format.
And vice versa if it's common for you to use 31 January, 2022 date format and see the time as 16:00 then your option is European format.

Members

You can manage your team members in your account settings. Please read more about it.

Billing info

You can edit your payment method and upgrade your plan on the account page.

---

title: "Members" description: ""

metadataTitle: ""

:::note This page is about Adapty dashboard members

If you want to give different access levels to users of your app check Access Level. :::

The Adapty dashboard members system allows you to grant different levels of access to Adapty and specify applications for each member.

Roles

The following roles are available for members in the Adapty dashboard:

Frame1434.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Owner: The Owner is the original creator of the Adapty account and holds the highest level of access and control. Owners have complete access to Adapty billing, allowing them to manage payment information and subscription plans. Additionally, only Owners and Admins can specify application access for new members. There can be only one Owner for each Adapty account.

Admin: Members with the Admin role have full access to the chosen applications. They can perform various management tasks, including creating and modifying paywalls, conducting A/B tests, analyzing analytics, and managing members within those applications.

Viewer: Members with the Viewer role have read-only access to the chosen applications. They can view information but cannot create or modify paywalls, A/B tests, and other features, invite new users, create new apps, and change the app settings.

Support: Members with the Support role have access only to user profiles in chosen applications. However, they cannot perform actions like adding new members or accessing any other sections of Adapty. This role is particularly suitable for support teams or individuals who need to assist customers with subscription-related inquiries or troubleshooting.

How to add a member

To access the members section and add new members, please navigate to the Account section in the Adapty dashboard. Within this section, you have the ability to select roles and specify apps for the new members, provided you have sufficient rights.

2023-06-08181614*Jun*08*2023*0619_PM.gif').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note It is only possible to invite an email that is not yet registered in Adapty. If your colleague already created a standalone account, invite another email address of theirs or contact Adapty support - we'll delete the problematic account. :::

If you want to transfer ownership of Adapty account, contact support.

By following these steps and utilizing the information provided, you can effectively manage member access and permissions within your Adapty account using the Adapty dashboard members system. For details on the number of members allowed for each plan, please refer to our Pricing documentation.

---

title: "Apple Platform resources" description: ""

metadataTitle: ""

Adapty offers SDKs and integrations tailored for Apple Platforms, simplifying the development of in-app purchases, subscriptions, paywalls, and A/B tests.

Use the following resources to maximize the benefits Adapty provides for Google Platforms.

Initial configuration in Google Play Console

1. Generate In-App Purchase Key in App Store Connect

Products and offers configuration in Google Play Console

1. Product in App Store
2. Offers in App Store

Additional information

1. Apple app privacy
2. Apple family sharing
3. App Store Small Business Program

---

title: "Product in App Store" description: ""

metadataTitle: ""

This page provides guidance on creating a product in App Store Connect. While this information may not directly pertain to Adapty's functionality, it serves as a valuable resource if you encounter challenges while creating products in your App Store Connect account.

To create a product that will be linked to Adapty:

1. Open **App Store Connect**. Proceed to **Monetization â†' Subscriptions** section in the left-side menu.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. If you haven't created a subscription group, click the **Create** button under the **Subscription Groups** title to initiate the process. Subscription Groups in App Store Connect categorize and manage your products, allowing users to switch between different offerings seamlessly. Note that it's not possible to create a subscription outside of a group.

3. In the opened **Create Subscription Group** window, enter a the new subscription group name in the **Reference Name** field. The reference name is a user-defined label or identifier that helps you distinguish and manage different subscription groups within your app.

   The reference name is not visible to users; it's primarily for your internal use and organization. It allows you to easily identify and refer to specific subscription groups when managing them within the App Store Connect interface. This can be particularly useful if you have multiple subscription offerings or want to categorize them in a way that makes sense for your app's structure.

subscriptiongroup.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Click the **Create** button to confirm the subscription group creation.

5. The subscription group is created and opened. Now you can create subscriptions in the group. Click the **Create** button under the **Subscriptions** title. If you add a new subscription to an existing group, then click a **Plus** button next to the **Subscriptions** title.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

6. In the opened **Create Subscription** window, enter its name in the **Reference Name** field and subscription unique code in the **Product ID** field.

   The Reference Name serves as an exclusive identifier within App Store Connect for your in-app subscription. It is not visible to your users on the App Store. We recommend using a clear, human-readable description that accurately represents the specific subscription you intend to create. Please note that this name must not exceed 64 characters in length.

   The Product ID is a unique alphanumeric identifier essential for accessing your product during the development phase and synchronizing it with Adapty, a service designed to manage in-app subscriptions. Only alphanumeric characters, periods, and underscores are allowed in the Product ID.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

7. Click the **Create** button to confirm the subscription creation.

8. The subscription is created and opened. Now select the duration of the subscription in the **Subscription Duration** list. Even if the subscription duration is already indicated in the subscription name, remember to complete the **Subscription Duration** field.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

9. Now it's time to set up the subscription price. To do so, click the **Add Subscription Price** button under the Subscription Prices title. You may need to scroll down to find them.

10. In the opened **Subscription Price** window, select the basic country in the **Country or Region** list and and basic currency in the **Price** list. Later Apple will automatically calculate the prices for all 175 countries or regions based on this basic price and the most recent foreign exchange rates.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

11. Click the **Next** button. In the opened **Price by Country or Region** window, you see the automatically recalculated prices for all countries. You can change them if you want.

bycountry.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

12. After updating regional prices, proceed by clicking the **Next** button at the bottom of the window.

13. In the opened **Confirm Subscription Price?** window, carefully review the final prices. To correct the prices, you can click the **Back** button to return to the **Price by Country or Region** window and update them. When you are ok with the prices, click the **Confirm** button.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

14. After closing the **Confirm Subscription Price?** window, remember to click the **Save** button in your subscription window. Without it, the subscription won't be created, and all entered data will be lost.

Please consider, that the steps provided so far focus on configuring an Auto-Renewable Subscription. However, if you intend to set up other types of in-app purchases, you can click on the **In-App Purchases** tab in the sidebar, instead of "Subscriptions." This will lead you to the section where you can manage and create various types of in-app purchases.

width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Add products to Adapty

Once you have completed adding your in-app purchases, subscriptions, and offers in App Store Connect, the next step is to add these products to Adapty.

---

title: "Offers in App Store" description: ""

metadataTitle: ""

Offers in the App Store are special deals or discounts provided by these platforms for in-app purchases. Developers use offers to provide users with exciting promotions, like discounted prices, free trials, or bundled offers. These promotions help attract and keep users engaged, making the app experience more rewarding. By using these special incentives, developers can boost user interest and loyalty, contributing to the overall success of their apps.

With iOS 13 Apple released Promotional offers as a way to promote your users to subscribe. Adapty supports Subscription offers.

:::note To use promotional offers, you have to upload subscription key to Adapty dashboard, so Adapty can sign the offers. :::

2023-07-25at_15.14.112x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Please also consider, that introductory offers on iOS are applied automatically if the user is eligible.

To include a promotional offer or free trial for your product, navigate to the promotional offers tab after setting up the pricing. You will find a **+ icon** next to promotional offers, and click on it to begin the setup.

In the subsequent modal, you'll encounter various configuration screens:

1. **Promotional offer reference name and promotional offer identifier**: This will configure the name and ID of the offer.
2. **Type of promotional offer**: You'll be able to choose the type of promotional offer from Pay as you go, Pay up front, and Free options. Then choose the desired Duration from the dropdown for the selected option.
3. **Prices for the offer for each country**

You can check our documentation to learn how to configure the products in the App Store.

---

title: "Apple app privacy" description: ""

metadataTitle: ""

Apple requires a privacy disclosure for all new apps and app updates both in the **App Privacy** section of the App Store Connect and as the app manifest file. Adapty is a third-party dependency to your app, therefore youâ€™ll need to properly disclose the ways you are using Adapty in regards to user's data.

Apple app privacy manifest

The privacy manifest file, named `PrivacyInfo.xcprivacy`, describes what private data your app uses and why. You as every app owner must create a manifest file for your app. Additionally, if you're integrating any extra SDKs, ensure the manifest files for those of them included in the SDKs that require a privacy manifest and signature list are included. When you build your app, Xcode will take all these manifest files and merge them into one.

Even though Adapty isn't on the list of SDKs that require a privacy manifest and signature, versions 2.10.2 and higher of the Adapty SDK include it for your convenience. Make sure to update the SDK to get the manifest.

While Adapty doesn't require any data to be included in the manifest file also called app privacy report, if you're using Adapty's `customerUserId` for tracking, it's necessary to specify it in your manifest file like so:

1. Add a dictionary to the `NSPrivacyCollectedDataTypes` array in your privacy information file.
2. Add the `NSPrivacyCollectedDataType`, `NSPrivacyCollectedDataTypeLinked`, and `NSPrivacyCollectedDataTypeTracking` keys to the dictionary.
3. Add string `NSPrivacyCollectedDataTypeUserID` (identifier of the `UserID` data type in the List of data categories and types to be reported in the manifest file) for the `NSPrivacyCollectedDataType` key in your `NSPrivacyCollectedDataTypes` dictionary.
4. Add `true` for the `NSPrivacyCollectedDataTypeTracking` and `NSPrivacyCollectedDataTypeLinked` keys in your `NSPrivacyCollectedDataTypes` dictionary.
5. Use the `NSPrivacyCollectedDataTypePurposeProductPersonalization` string as the value for the `NSPrivacyCollectedDataTypePurposes` key in your `NSPrivacyCollectedDataTypes` dictionary.

If you target your paywalls to audiences with custom attributes, consider carefully what custom attributes you use and if they match the data categories and types to be reported in the manifest file. If so, repeat the steps above for every data type.

After you report all data types and categories you collect, create your app's privacy report as described in Apple documentation.

Apple app privacy disclosure in App Store Connect

In the **App Privacy** section of the App Store Connect, make sure to clearly explain how you're using Adapty in relation to user data.

Data types

âœ… = Required
ðŸ‘€ = May be required (see details below)
âŒ = Not required

| Data type | Required | Note | |---------|--------|----| Identifiers | âœ… |

If you are identifying users with a customerUserId, select 'User ID'.

Adapty collects IDFA, so you have to select 'Device ID'.

| | Purchases | âœ… | Adapty collects purchase history from users. | | Contact Info, including name, phone number, or email address | ðŸ‘€ | Required if you pass personal data like name, phone number, or email address using **`updateProfile`** method. | | Usage Data | ðŸ‘€ | If you are using analytics SDKs such as Amplitude, Mixpanel, AppMetrica, or Firebase, this may be required. | | Location | âŒ | Adapty does not collect precise location data. | | Health & Fitness | âŒ | Adapty does not collect health or fitness data from users. | | Sensitive Info | âŒ | Adapty does not collect sensitive information. | | User Content | âŒ | Adapty does not collect content from users. | | Diagnostics | âŒ | Adapty does not collect device diagnostic information. | | Browsing History | âŒ | Adapty does not collect browsing history from users. | | Search History | âŒ | Adapty does not collect search history from users. | | Contacts | âŒ | Adapty does not collect contact lists from users. | | Financial Info | âŒ | Adapty does not collect financial info from users. |

Required data types

Purchases

When using Adapty, you must disclose that your app collects â€˜Purchasesâ€™ information.

2023-08-25at_12.32.552x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Identifiers

If you are identifying users with **`customerUserId`**, you'll need to select 'User ID'.

Adapty collects IDFA, so you'll need to select 'Device ID'.

2023-08-25at_12.35.272x.png').default} style={{ border: 'none', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

After making your selections, you'll need to indicate how the data is used similar to the Purchases section.

After making your privacy selections, Apple will show a preview of your app's privacy section. If you have chosen Purchases and Identifiers as described above, your app's privacy details should look something like this:

2023-08-25at_12.36.442x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

---

title: "Apple family sharing" description: ""

metadataTitle: ""

Apple's family sharing enables the distribution of in-app purchases among family members, offering users of group-oriented apps, such as video streaming services and kids' apps, a convenient way to split subscriptions without having to share their Apple ID. By allowing up to five family members to utilize a subscription, Family sharing can potentially improve customer engagement and retention for your app.

In this guide, we will provide instructions on how to opt-in subscriptions to Family Sharing and explain how Adapty manages purchases that are shared within a family.

To get started with enabling Family Sharing for a particular product, head over to App Store Connect. Family Sharing is turned off by default for both new and existing in-app purchases, so it is necessary to enable it individually for each in-app purchase. You can easily do this by accessing your **app's page,** navigating to the corresponding in-app purchase page, and selecting the **Turn On** option in the Family Sharing section.

Keep in mind that once you enable Family Sharing for a product, **it cannot be turned off again**, as this would disrupt the user experience for those who have already shared the subscription with their family members. Also, please consider that, only non-consumables and subscriptions can be shared.

2023-03-28at_17.15.342x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

On the displayed modal simply click on the **Confirm** button to finalize the setup process. After doing so, the Family Sharing section should update to display the message, "This subscription can be shared by everyone in a family group." This confirms that the subscription is now enabled for Family Sharing and can be shared among up to five family members.

Adapty makes it easy to support Family Sharing without any additional effort required. You just need to simply configure your products from the App Store, and once you **enable** it from App Store Connect **Family Sharing** will be automatically available in **Adapty**, that will be received as an event on the webhook.

:::note Please note that Family Sharing is not supported in the sandbox environment. :::

One thing you can consider is that when a user purchases a subscription and shares it with their family members, there is a **delay of up to one hour** before it becomes available to them. Apple designed this delay to give the user time to change their mind and undo the sharing if they want to. However, if the subscription is renewed, there is no delay in making it available to the family members.

When a user purchases a Family Shareable in-app product, the transaction will appear in their receipt as usual, but with the addition of a new field called `in_app_ownership_type` with the value `PURCHASED`. Furthermore, a new transaction will be created for all family members, which will have a different `web_order_line_item_id` and `original_transaction_id` compared to the original purchase, as well as an `in_app_ownership_type` field with the value `FAMILY_SHARED`.

To ensure accurate revenue calculation, on the Adapty side, only transactions with an `in_app_ownership_type` value of `PURCHASED` are considered. This means that we don't take into account `FAMILY_SHARED` values in analytics and do not send events based on them.

To identify the other family members on Adapty, you can find them in the event details. First, locate the original family purchase transaction. Then, examine the event details for that transaction, specifically looking for the same product, purchase date, and expiration date. By analyzing the event details, you can identify other family membership transactions associated with the original purchase.

---

title: "App Store Small Business Program" description: ""

metadataTitle: ""

Learn about how Adapty calculates proceeds for both the App Store and Google Play Store, taking into account the reduced commission rate offered by the Small Business Program. Also, you can check the instructions on how to manage your Small Business Program membership status for the App Store in the Adapty Dashboard. By keeping your membership status up to date, you can ensure that Adapty accurately calculates your sales commission and provides reliable information on your transactions.

Adapty also supports the reduced service fee program for Google Play. You can reference this document for more details.

App Store Small Business Program

The App Store Small Business Program is a scheme that reduces the commission on App Store sales for eligible small businesses from 30% to 15%. The program is available to developers who earn less than $1 million in annual App Store revenue, subject to certain eligibility criteria. Further information about the program can be found on the App Store Small Business Program page.

Acknowledging your membership in the Small Business Program in your app settings is essential, as the reduced commission rate will impact the data sent for integrations and the information displayed in Adapty's charts. By providing this information through Adapty, you can simplify the enrollment process and take advantage of the reduced commission rate offered by the program.

To join the App Store Small Business Program, the first step is to visit the Apple Developer website. Before applying, please make sure that you are the Account Holder in the Apple Developer Program, have accepted the latest Paid Applications contract in App Store Connect, and can list all associated developer accounts to the account for which you are applying.

After reviewing the program's terms, click the 'Enroll' button and sign in to your Apple Developer account. Apple's enrollment form will automatically fill in information like your name, email, and Team ID. After submitting your enrollment form, Apple will review your application. Once your enrollment is processed, you will receive a confirmation email that your enrollment is being reviewed.

How Adapty calculates the proceeds App Store

Adapty can accurately calculate your app's earnings by deducting Apple's and Google's commissions and taking into account your eligibility for the Small Business Program.

In the Adapty Dashboard, the Small Business Program membership status is assigned to each individual app based on the developer's representation of multiple apps from different companies in their account. This means

that the eligibility for the program is determined on a per-app basis.

You can add multiple periods by selecting a range of dates for each period in the same field. The date range represents the start and end date of the period during which your business was a member of the Small Business Program. Please note that the Entry Date refers to the earliest date in the range when your business became a member of the program, and the Exit Date refers to the latest date in the range when your business officially left or was removed from the program.

You can select your entry date according to your preference. However, it's important to note that if you select a past date, any webhooks and integration events already processed will not be resent with corrected pricing data. To ensure the accuracy of pricing data sent to your integrations, it's advisable to set your effective entry date as soon as possible. This way, you can receive reliable and up-to-date information on your transactions and make informed decisions accordingly.

Letting Adapty know

To manage your Small Business Program membership status for the App Store, go to the **App Settings > General tab** in your Adapty account. Click the **Add period** button to specify your membership status for a specific period range. In the "Period" field, select a date range that indicates your business's membership start and end dates. This range can include any date in the past or the future.

You can add additional membership periods by clicking on the "Add Period" button again.

2023-04-11at_15.00.482x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can select your period start according to your preference. However, if you select a past period start, any webhooks and integration events already processed will not be resent with corrected pricing data. To ensure the accuracy of pricing data sent to your integrations, it's advisable to set your effective period start as soon as possible. This way, you can receive reliable and up-to-date information on your transactions and make informed decisions accordingly.

Please note that the Small Business Program membership status will only apply to the specific period range you've specified. Once the period end is reached, you'll need to add another period if you want to continue with the Small Business Program membership status.

To ensure that we calculate your sales commission correctly, please enter the effective exit date in your Adapty Dashboard app settings as soon as possible if your business has left the Small Business Program. If no exit date is provided, we will continue to calculate your commission based on the reduced rate.

---

title: "Google Platform resources" description: ""

metadataTitle: ""

Adapty offers SDKs and integrations tailored for Google Platforms, simplifying the development of in-app purchases, subscriptions, paywalls, and A/B tests.

Use the following resources to maximize the benefits Adapty provides for Google Platforms.

Initial configuration in Google Play Console

1. Enable Developer APIs in Google Play Console
2. Create a service account in the Google Cloud Console
3. Generate service account key file
4. Grant permissions to a service account in the Google Play Console

Products and offers configuration in Google Play Console

1. Creating products for your mobile app
2. Creating offers to products

Additional information

1. Google Play Data Safety
2. Apple app privacy
3. Google Reduced Service Fee

---

title: "Product in Play Store" description: ""

metadataTitle: ""

This page provides guidance on creating a product in Play Store. While this information may not directly pertain to Adapty's functionality, it serves as a valuable resource if you encounter challenges while creating products in Google Play console.

Product refers to a digital item or service that you offer within your app in Play Store, typically available for purchase. These can include in-app products such as one-time purchases, subscriptions, or other digital goods that users can acquire while using your application.

In the Googleâ€™s billing system, subscriptions can incorporate multiple base plans, each providing various discounts or offers. This structure is comprised of three main components:

- **Subscriptions:** These represent sets of benefits that users can enjoy for a specific period (the items being sold). For instance, a "Gold tier" providing premium features for subscribers.
- **Base plans:** These represent specific configurations of billing periods, renewal types, and prices (how the items are sold). Examples include "annual with auto-renewal" or "prepaid monthly."
- **Offers:** These entail discounts available to eligible users, modifying the base plan's price. For instance, a "free 14-day trial for new users."

:::note Adapty SDK 2.6.0 already supports Google Billing Library v5 and v6 as well as the modern Google Play subscriptions structure such as multiple base plans per subscription, multiple offers per base plan, and multiple phases per offer.

Please refer to our documentation for more details about how to migrate to Adapty SDK 2.6.0. :::

How to create a product in Play Store?

Product refers to a digital item or service that you offer within your app, typically available for purchase. These can include in-app products such as one-time purchases, subscriptions, or other digital goods that users can acquire while using your application.

To set up a product for Android devices:

1. Open **Monetize -> Subscriptions** or **Monetize -> In-app products** section in the left menu of the Google Play Console.

/ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

2. Click the **Create subscription** button.

subscriptionGP.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

3. In the opened **Create subscription** window, enter the subscription ID in the **Product ID** field and the subscription name in the **Name** field.

   Product ID has to be unique and must start with a number or lowercase letter, and can also contain underscores (_), and periods (.). It is used to access your product during development and synchronize it with Adapty. Once a Product ID is assigned to a product in the Google Play Console, it cannot be reused for any other apps, even if the product is deleted.

   When naming your product ID, it is advisable to follow a standardized format. We recommend using a more concise approach and naming the product `<subscription name>.<access level>`. Then, you can control the duration and billing frequency through the use of base plans, such as weekly, monthly, and so on.

   The Name is used for your reference only, it will be visible on your Google Play Store listing, so feel free to use any descriptive name you need. It is limited to 55 characters.

4. Click the **Create** button to confirm the subscription creation.

:::note Google Play subscription products in Adapty

Adapty products correspond to Base Plans for Google Play subscriptions since those are the products available for customers to purchase. Adapty seamlessly handles the migration of existing Google Play subscriptions along with their corresponding base plans in products, requiring no additional action from you. However, when you add a new product in Adapty, you will be responsible for providing both the base plan ID and the product ID. :::

Create a base plan

For subscription products, you'll need to add a base plan. Base plans determine the billing period, price, and renewal type for customers to purchase your subscription. Please note that customers do not directly purchase a subscription product. Instead, they always buy a base plan within a subscription.

To create a base plan:

1. Open **Monetize -> Subscriptions** section in the left menu of the Google Play Console. Once there, locate the subscription to which you'd like to add a base plan.
2. Click the **View subscription** button next to the subscription.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. After the subscription details open. click on the **Add base plan** button under the **Base plans and offers** title. You may need to scroll down to find it.

baseplan.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

4. In the opened **Add base plan** window, enter a unique identifier for the base plan in the Base **Plan ID** field. It must start with a number or lowercase letter, and can contain numbers (0-9), lowercase letters (a-z) and hyphens (-). and complete the required fields.

2023-07-20at_16.51.412x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

5. Specify the prices per region.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

6. Đ¡lick the **Save** button to finalize the setup.
7. Đ¡lick the **Activate** button to make the baseline active.

Keep in mind that subscription products can only have a single base plan with consistent duration and renewal type in Adapty.

Fallback products

:::warning Support for non backwards-compatible base plans

Older versions of Adapty SDKs do not support Google Billing Library v5+ features, specifically multiple base plans per subscription product and offers. Only base plans marked as**backwards compatible** in the Google Play Console are accessible with these SDK versions. Note that only one base plan per subscription can be marked as backwards compatible. :::

2023-07-20at_17.03.252x.png').default} style={{ border: '1px solid #727272', /* border width and color / *width: '700px',* / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

To fully leverage the enhanced Google subscription configurations and features in Adapty, we offer the capability to set up a backward compatible fallback product. This fallback product is exclusively utilized for apps using older versions of the Adapty SDK. When creating Google Play products, you now have the option to indicate whether the product should be marked as backward compatible in the Play Console. Adapty utilizes this information to determine whether the product can be purchased by older versions of the SDK (versions 2.5 and below).

Suppose you have a subscription named `subscription.premium` that offers two base plans: weekly (backward compatible) and monthly. If you add `subscription.premium:weekly` product to Adapty, you don't need to indicate a backward compatible product. However, in the case of `subscription.premium:monthly` product, you will need to specify a backward compatible product. Failing to do so could result in an unintended purchase of `subscription.premium:weekly` product in Google 4th billing library. To address this scenario, you should create a separate product where the base plan is also monthly and marked as backward compatible. This ensures that users who select the `subscription.premium:monthly` option will be billed correctly at the intended frequency.

Add products to Adapty

Once you have completed adding your in-app purchases, subscriptions, and offers in App Store Connect, the next step is to add these products to Adapty.

---

title: "Offers in Google Play" description: ""

metadataTitle: ""

With Billing Library v5, Google introduced a new way of working with offers. It gives you much more flexibility, but it's important to configure them properly. After reading this short guide from Adapty, you'll have a full understanding of Google Play Offers.

:::note Checklist to successfully use Google Play offers

1. Create and activate offers in Google Play Console.
2. Add offers to Adapty Products.
3. Choose the offer to use in Adapty Paywall.
4. Use Adapty SDK 2.6 or newer.
5. Check eligibility criteria for the offer in Google Play Console if everything is configured, but the offer is not applied. :::

Overview

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

Before Google Play Billing Library v5 a subscription could only have one offer. If you wanted to test different offers, for example, a 3-day free trial vs a 1-week free trial, you would have to create 2 different subscriptions, which is not optimal.

Now you can create multiple offers for every base plan (previously known as subscription) and this means that you have to decide which offer should be used at a given moment. Please check the docs on base plans if you're not familiar with them.

Configuring offers in Google Play

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

In the screenshot above, you can see a subscription `premium_access`(1) with two base plans: `1-month` (2) and `1-year` (3). Offers are always created for base plans.

1. To create an offer, click **Add offer** and choose the base plan from the list.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

2. Enter the offer ID. It will be later used in the analytics and Adapty dashboard, so give it a meaningful name.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

3. Choose the eligibility criteria:
    1. **New customer acquisition**: the offer will be available only to new subscribers if they haven't used this offer in the past. This is the most common option and should be used by default.
    2. **Upgrade**: this offer will be available for the customers upgrading from the other subscription. Use it when you want to promote more expensive plans to your existing subscribers, for example, customers upgrading from the bronze to the gold tier of your subscription.
    3. **Developer determined**: you can control who can use this offer from the app code. Be cautious using it in production to avoid possible fraud: customers can activate a free or discounted subscription over and over again. A good use case for this offer type is winning back churned subscribers.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

4. Add up to two pricing phases to your offer. There are three phase types available:
    1. **Free trial**: the subscription can be used for free for a configured amount of time (minimum 3 days). This is the most common offer.
    2. **Single payment**: the subscription is cheaper if the customers pay upfront. For example, normally a monthly plan costs $9.99, but with this offer type, the first three months cost $19.99, a 30% discount.
    3. **Discounted recurring payment**: the subscription is cheaper for the first `n` periods. For example, normally a monthly plan costs $9.99, but with this offer type, each of the first three months costs $4.99, a 50% discount.
    An offer can have two phases. In this case, the first phase must be a Free trial, and the second one is either a Single payment or a Discounted recurring payment. They would be applied in this order.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

5. Activate the offer to use it in the app.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

After activating the offer, you should copy its ID to use in Adapty.

./ width: '700px', / image width / *display: 'block',* / for alignment / *margin: '0 auto'* / center alignment */ } } />

The `1-month` plan has three offers: `free-trial-1-week` (1), `free-trial-3-days` (2), `pay-up-front-3months-30p` (3). The `1-year` plan has one offer: `free-trial-1-week` (1). As you can see, offer IDs can be the same for different base plans.

Adding offers to Adapty products

Let's create a 1-month product in Adapty with all the offers. You can do it from a single screen.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

1. Choose the name, access level, and period.
2. Copy the Product ID and Base plan ID from Google Play Console and paste them into the corresponding fields in Adapty.
3. Copy an offer ID from Google Play Console and paste it into the Google Play Offer ID field in Adapty. Provide a user-friendly name for the offer. If you have multiple offers, add all of them by clicking **Add offer**.
4. Save the changes.

Choosing the offer in Adapty paywalls

Finally, you have to choose, which offer should be displayed on the given paywall.

When creating a paywall or editing a draft of the paywall, choose the offer from the dropdown next to the product. This offer will be then used during the purchase from the paywall if the customer is eligible for the offer in the first place.

If you configure a paywall like this, a monthly subscription will not have a free trial. A yearly subscription will have a 1-week trial if the customer is eligible.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

:::note If you can't edit the products on the paywall, it means that the paywall is not in the draft state. You can duplicate it or create a new paywall, and then select the new paywall in the placement. :::

---

title: "Google Play Data Safety" description: ""

metadataTitle: ""

The Data Safety section available on Google Play provides a simple method for app developers to inform users about the data collected or shared by their app, as well as highlight their app's critical privacy and security measures. This information enables users to make more informed decisions when selecting which apps to download and use.

Here is a short guide on data that Adapty collects to help you provide the required information to Google Play.

Data Collection and Security

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Does your app collect or share any of the required user data types?
Select 'Yes' as Adapty collects a customer's purchase history.

Is all of the user data collected by your app encrypted in transit?
Select 'Yes' as Adapty encrypts data in transit.

Do you provide a way for users to request that their data is deleted?
If selecting 'Yes', ensure your customers have a way to contact your support team to request a data deletion. You will be able to delete the customer directly from the Adapty dashboard or via REST API.

Data Types

Here is a list of the data types that Google requires for reporting, and we have specified whether Adapty collects any particular type of data.

| Data Type | Details | | :---------- | :---------- | | Location | Is not collected by Adap
Health and Fitness | Is not collected by Adapty | | Photos and Videos | Is not collected by Adapty | | Files and Docs | Is not collected by Adapty | | Calendar | Is not collected by Adapty | | Contacts | Is not collected by Adapty | | User Content | Is not collected by Adapty | | Browsing History | Is not collected by Adapty | | Search History | Is not collected by Adapty | | App Info and Performance | Is not collected by Adapty | | Web Browsing | Is not collected by Adapty | | Contact Info | Is not collected by Adapty | | Financial Info | Adapty collects purchase history from users | | Personal Info and Identifiers | Adapty collects User ID and some other identifiable contact information including name, email address, phone number, etc, if you explicitly pass them to Adapty SDK. | | Device and other identifiers | Adapty collects data on device id. |

Data usage and handling

User IDs

1. Is this data collected, shared, or both?
This data is collected by Adapty. If you are using integrations set up between Adapty and third parties that are not considered service providers, you may need to disclose "Shared" here as well.
2. Is this data processed ephemerally?
Select 'No'.
3. Is this data required for your app, or can users choose whether it's collected?
This data collection is required and cannot be turned off.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

4. Why is this user data collected? / Why is this user data shared?
Select the 'App functionality' and 'Analytics' checkboxes.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Financial Info

If you are using Adapty, you must disclose that your app collects 'Purchase history' information from the Data types section in Google Play Console.

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Device or other IDs

2023-03-01at_17.55.312x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

Next Steps

Once you have made your data safety selections, Google will display a preview of your app's privacy section. If you have opted for "Financial Info" and "Device or other IDs" as mentioned earlier, your privacy information should appear similar to the following example

./ width: '700px', / image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

If you are prepared to submit your app for App Review, please refer to our Release Checklist document for further guidance on preparing your app for submission.

---

title: "Google Reduced Service Fee" description: ""

metadataTitle: ""

Learn how Adapty can help you manage your proceeds on Google Play Store, including the reduced service fee program for developers earning less than $1,000,000 USD annually. By following the necessary steps to join the program and updating your membership status in the Adapty Sashboard, you can ensure that your sales commission is accurately calculated, and you receive reliable information on your transactions.
Adapty also supports the Small Business Program for App Store. You can reference this document for more details.

Google's 15% Reduced Service Fee

Developers who earn less than $1,000,000 USD annually are eligible to join a reduced service fee tier offered by Google. Under this tier, the commission fee is lowered to 15% instead of the standard rate of 30%.

Developers offering automatically renewing subscription products are eligible for a reduced service fee of 15%, independent of their participation in other programs offered by Google Play. You can read more about the service fees here.

To participate in the Google Play Reduced Service Fee program, you must have a payment profile and create an Account Group where your Developer Account is the Primary Developer Account. You also need to inform Google if you have any Associated Developer Accounts (ADAs) and link your account to the group. Once these steps are completed, you will automatically be enrolled in the program, and you'll be eligible for the reduced commission rate on your first $1,000,000 USD in revenue per year. For more detailed information on the necessary steps, please refer to Google's documentation.

How Adapty calculates the proceeds Play Store

Adapty can accurately calculate your app's earnings by deducting Google's commissions and taking into account your eligibility for the Reduced Service Fee program.

In the Adapty Dashboard, the Reduced Service Fee membership status is assigned to each individual app based on the developer's representation of multiple apps from different companies in their account. This means that the eligibility for the program is determined on a per-app basis.

You can add multiple periods by selecting a range of dates for each period in the same field. The date range represents the start and end date of the period during which your business was a member of the Small Business Program. Please note that the Entry Date refers to the earliest date in the range when your business became a member of the program, and the Exit Date refers to the latest date in the range when your business officially left or was removed from the program.

You can select your entry date according to your preference. However, it's important to note that if you select a past date, any webhooks and integration events already processed will not be resent with corrected pricing data. To ensure the accuracy of pricing data sent to your integrations, it's advisable to set your effective entry date as soon as possible. This way, you can receive reliable and up-to-date information on your transactions and make informed decisions accordingly.

Letting Adapty know

To manage your Reduced Service Fee membership status for Google Play, go to the **App Settings > General tab** in your Adapty account. Click the **Add period** button to specify your membership status for a specific period range. In the "Period" field, select a date range that indicates your business's membership start and end dates. This range can include any date in the past or the future. You can add additional membership periods by clicking on the "Add Period" button again.

2023-04-11at_15.00.482x.png').default} style={{ border: '1px solid #727272', /* border width and color */ width: '700px', /* image width */ display: 'block', /* for alignment */ margin: '0 auto' /* center alignment */ }} />

You can select your period start according to your preference. However, if you select a past period start, any webhooks and integration events already processed will not be resent with corrected pricing data. To ensure the accuracy of pricing data sent to your integrations, it's advisable to set your effective period start as soon as possible. This way, you can receive reliable and up-to-date information on your transactions and make informed decisions accordingly.

Please note that the Reduced Service Fee membership status will only apply to the specific period range you've specified. Once the period end is reached, you'll need to add another period if you want to continue with the Reduced Service Fee membership status.

To ensure that we calculate your sales commission correctly, please enter the exit date in the Adapty Dashboard app settings as soon as possible if your business has left the Reduced Service Fee. If no exit date is provided, we will continue to calculate your commission based on the reduced rate.

---

title: "Firebase apps" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

This page is about integration of Adapty in you app, if it works on Firebase.

:::note Get started

This is not all steps required for Adapty to work, just some useful tips for integration with Firebase. If you want to integrate Adapty in you app, you should read Quickstart Guide first :::

User Identification

If you're using Firebase auth, this snippet may help you keep your users in sync between Firebase and Adapty. Note that it's just an example, and you should consider your app auth specifics.

```swift import Adapty import Firebase

@UIApplicationMain class AppDelegate: UIResponder, UIApplicationDelegate {

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    // Configure Adapty before Firebase
    Adapty.activate("YOUR_API_KEY")
    Adapty.delegate = self

    // Configure Firebase
    FirebaseApp.configure()

    // Add state change listener for Firebase Authentication
    Auth.auth().addStateDidChangeListener { (auth, user) in
        if let uid = user?.uid {
            // identify Adapty SDK with new Firebase user
            Adapty.identify(uid) { error in
                if let e = error {
                    print("Sign in error: \(e.localizedDescription)")
                } else {
                    print("User \(uid) signed in")
                }
            }
        }
    }

    return true
}

}

extension AppDelegate: AdaptyDelegate {

// MARK: - Adapty delegate
func didReceiveUpdatedPurchaserInfo(_ purchaserInfo: PurchaserInfoModel) {
    // You can optionally post to the notification center whenever
    // purchaser info changes.

    // You can subscribe to this notification throughout your app
    // to refresh tableViews or change the UI based on the user's
    // subscription status

    NotificationCenter.default.post(name: NSNotification.Name(rawValue: "com.Adapty.PurchaserInfoUpdatedNotification"), object: purchaserInfo)
}

} </TabItem> <TabItem value="kotlin" label="Android with Firebase" default> kotlin class App : Application() {

override fun onCreate() {
    super.onCreate()
    // Configure Adapty
    Adapty.activate(this, "YOUR_API_KEY")

    Adapty.setOnPurchaserInfoUpdatedListener(object : OnPurchaserInfoUpdatedListener {
        override fun onPurchaserInfoReceived(purchaserInfo: PurchaserInfoModel) {
            // handle any changes to subscription state
        }
    })

    // Add state change listener for Firebase Authentication
    FirebaseAuth.getInstance().addAuthStateListener { auth ->
        val currentUserId = auth.currentUser?.uid

        if (currentUserId != null) {
            // identify Adapty SDK with new Firebase user
            Adapty.identify(currentUserId) { error ->
                if (error == null) {
                    //success
                }
```

```
            }
        } else {
            Adapty.logout { }
        }
    }
}

} ```

---

title: "Getting started with server-side API" description: ""

## metadataTitle: ""

With API you can:

1. Get user's subscription status.
2. Activate a subscription for a user with an access level.
3. Get user's attributes.
4. Set user's attributes.

:::note You can't get subscription events via API, but you can use Webhook or direct integration with a service that you're using. :::

To correctly work with API you need to use a unique ID for your users. This may be an email, a phone number, your internal ID. Without such an ID it's impossible to identify the same user on multiple platforms.

## Case 1: Syncing subscribers between web and mobile

Whenever Web payment providers you use such as Stripe, ChargeBee, or any other, you can sync subscribers. For that:

1. *Use a unique ID for your users*. For example, email or phone number.
2. Check subscription status via API.
3. If a user is freemium, show him a paywall on the Web.
4. After successful payment, update subscription status in Adapty via API.
5. Your subscribers will be automatically in sync with mobile.

## Case 2: Grant a subscription

:::note Due to security reasons, you can't grant a subscription via mobile SDK. :::

Imagine a case, when you run a promotional campaign with offers 7 days of a trial and you want to sync in with mobile experience. To do that:

1. Get a unique ID for a user.
2. Set premium access via paid access level with API with a duration of 7 days.

After 7 days users who won't subscribe will be downgraded to the free tier.

## Case 3: Syncing users' attributes and custom properties

You may have custom attributes for your users, other than defaults such as IDFA, device model, etc. For example, in a language learning service, you may want to save the number of words a student has learned. To do that:

1. Get a unique ID for a user.
2. Update attribute with API or SDK.

With such attributes you can, for example, create a segment and run an A/B test.

To learn more about S2S API go to API Specs.

---

title: "API specs" description: ""

## metadataTitle: ""

**Base URL**: `https://api.adapty.io/api/v1/sdk`

## Authorization

Each API request must be signed with the Secret Key.

When calling API:

- You must set **Authorization** header with value "Api-Key {secret_token}" (without quotes) to each request, for example `Api-Key secret_live_BEHrYLTr.ce5zuDEWz061FRNiaJC8mrLtL8fUwswD`
- Use JSON payload in the request body for POST and PATCH requests
- All request must set header **Content-Type**: application/json

## Working with customer user ID

Most server-side API requests allow passing `customer_user_id` as a URL parameter. This makes it easy for you to query/update data in Adapty, without having to store Adapty's `profile_id`. In most cases, you should pass `customer_user_id` as is, without any modifications.

However, if your `customer_user_id` contains reserved URI characters, for example `/`, `?`, `+` you should pass it encoded. Use the Base64URL encoding (not the regular Base64). This way, all special characters will be encoded and Adapty will decode it upon receiving. To tell Adapty the `customer_user_id` is encoded, pass the `is_user_id_base64url_encoded=1` get parameter. Note, that passing the `is_user_id_base64url_encoded=1` get parameter without actual encoding, will end up with 400 validation error.

You should only encode the `customer_user_id` if you pass it as a URL path. When sending `customer_user_id` inside the JSON payload (for example, when creating the profile), you should not encode it.

```python title="Python"

## Don't encode

customer*user*id = '123' # GET: /profiles/123/ customer*user*id = 'abc' # GET: /profiles/abc/ customer*user*id = '3c410419-9959-447a-84b5-be7cb6a308d9' # GET: /profiles/3c410419-9959-447a-84b5-be7cb6a308d9/

## Base64URL encode

customer*user*id = '123+456' # GET: /profiles/MTIzKzQ1Ng==/?is*user*id*base64url*encoded=1 customer*user*id = 'abc/def' # GET: /profiles/YWJjL2RlZg==/?is*user*id*base64url*encoded=1 customer*user*id = '012?012' # GET: /profiles/MDEyPzAxMg==/?is*user*id*base64url*encoded=1 ```

## Requests

### Prolong/grant a subscription for a user

`text title="Text" POST: /profiles/{profile_id_or_customer_user_id}/paid-access-levels/{access_level}/grant/`

Path parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **profile*id*or*customer*user*id*** | str | âœ… | âŒ | Adapty profile ID o developer's internal ID | | **accesslevel** | str | âœ… | âŒ | ID (slug) of a paid access level. Find it in Adapty Dashboard |

Request parameters:

| Param | Type | Required | Nullable | Description | |-----|----|--------|--------|----------| **expires*at*** | ISO 8601 date | âœ…* see below | âŒ | Subscription deadline | | **durationdays** | int | âœ…* see below | âŒ | Additional days to a current subscription** | | **is*lifetime*** | bool | âœ…* see below | âŒ | If set true, then a user will forever have a paid access level forever | | **startsat** | ISO 8601 date | âŒ | âŒ | If the start time of the action is in the future, then you can transfer it. If the start time and the period are indicated, the period will be counted from the indicated time | | **vendor*product*id** | str | âŒ | âŒ | Vendor product ID which initiates subscription renewal. The

default value is **adapty_promotion** | | **base*plan*id** | str | âœ… | âœ… | Base plan ID in the Google Play Store or price ID in Stripe. | | **vendor*original*transaction*id*** | str | âŒ | âŒ | ID of the original transaction in the subscription renewal chain in a vendor environment. | | **vendortransaction*id*** | str | âŒ | âŒ |

Transaction ID in a vendor environment.

If it is the same as **vendororiginal*transaction*id** or if **vendor*original*transaction*id*** is absent, Adapty considers it the first subscription purchase. If it differs from **vendororiginal*transaction*id**, Adapty considers the purchase the subscription renewal.

| | **store** | str | âŒ | âŒ | A store where users purchased a product, such as **app_store** and **play_store**, can be custom. Default is **adapty** | | **introductory*offer*type** | str | âŒ | âŒ | A type of introduction offer. Available values are **free_trial**, **pay_as_you_go**, and **pay_up_front**. | | **price** | float | âŒ | âŒ |

Price of the subscription/purchase to save in transaction.

The first subscription purchase with a zero price is considered a free trial, while a renewal with a zero price is considered a free subscription renewal.

If you provide price, provide `price_locale` as well.

| | **price*locale*** | str | âŒ | âŒ | The currency of the transaction in the three-letter format. `USD` is used by default. | | **proceeds** | float | âŒ | âŒ | Proceeds (price that is reduced due to stores' fee) of the subscription/purchase to save in transaction. | | **issandbox** | bool | âŒ | âŒ | Boolean indicating whether the product was purchased in the sandbox or production environment. |

**Paid access level**

There are three ways to grant users a subscription. So, at least one of **is_lifetime**, **expires_at**, or **duration_days** must be set. If more than one param is set, then **is_lifetime=true** has a maximum priority, then **expires_at**, and lastly **duration_days**.

As all payment processing is done by Apple/Google, Adapty can not control or affect it. So, when using **duration_days** to a current subscription, remember that a user still will be charged on a needed day. For example, the user has a monthly subscription and the next charge date will be the 5th of April. You grant a user additional 7 days, *but the user still is charged on the 5th of April!*. It's best using **duration_days** with never subscribed users or churned. In that case, reference day is a day of granting.

**Transaction**

If all **vendor*product*id**, **vendor_transaction_id,** and **store** are specified, Adapty creates and saves transaction entry so this grant is accounted in Charts (Revenue, MRR, Subscriptions) unless the transaction with these parameters already exists (e.g. it was generated by iOS purchase). If **price** is specified, it is associated with this transaction.

Currently, this type of transaction does not generate any profile events, does not affect users' subscription status, and does not show up in the **Event Feed**.

Also, be aware that these transactions affect billing since they are counted towards MTR.

Sample request:

```
json title="Json" { "starts_at": "2020-01-15T15:10:36.517975+0000", "expires_at": "2020-02-15T15:10:36.517975+0000", "vendor_product_id": "basic_subscription_1_month",
"vendor_transaction_id": "1000000630116569", "store": "app_store" }
```

Sample response:

```
json title="Json" { "data": { "app_id": "ff90dd2e-e7f2-454b-9d86-071036a284fe", "profile_id": "77112400-89f1-4465-b9c9-5437e58c6688", "customer_user_id": "iwitaly@adapty.io",
"paid_access_levels": { "premium": { "id": "premium", "is_active": true, "is_lifetime": false, "expires_at": "2023-03-29T15:30:34.000000+0000", "starts_at": null, "will_renew":
false, "vendor_product_id": "adapty_promotion", "base_plan_id": "premium_autorenewing", "vendor_transaction_id": "1000000630116569", "vendor_original_transaction_id":
"1000000625263604", "store": "adapty", "activated_at": "2020-03-26T16:24:19.497674+0000", "renewed_at": "2020-03-26T16:24:19.497674+0000", "unsubscribed_at": null,
"billing_issue_detected_at": null, "is_in_grace_period": false, "active_introductory_offer_type": "free_trial", "active_promotional_offer_type": null, "active_promotional_offer_id":
null, "cancellation_reason": null } }, "subscriptions": { "com.adapty.premium.monthly": { "is_active": false, "is_lifetime": false, "expires_at": "2020-02-21T16:30:34.000000+0000",
"starts_at": null, "will_renew": false, "vendor_product_id": "com.adapty.premium.monthly", "base_plan_id": "monthly_autorenewing", "vendor_transaction_id": "1000000630116569",
"vendor_original_transaction_id": "1000000625263604", "store": "app_store", "activated_at": "2020-02-10T19:14:02.000000+0000", "renewed_at": "2020-02-21T16:25:34.000000+0000",
"unsubscribed_at": "2020-02-21T16:30:34.000000+0000", "billing_issue_detected_at": "2020-02-21T16:30:34.000000+0000", "is_in_grace_period": false, "active_introductory_offer_type":
null, "active_promotional_offer_type": null, "active_promotional_offer_id": null, "cancellation_reason": "voluntarily_cancelled", "is_sandbox": true }, "com.adapty.premium.weekly":
{ "is_active": false, "is_lifetime": false, "expires_at": "2020-02-10T19:32:00.000000+0000", "starts_at": null, "will_renew": true, "vendor_product_id": "com.adapty.premium.weekly",
"base_plan_id": "weekly_autorenewing", "vendor_transaction_id": "1000000625265713", "vendor_original_transaction_id": "1000000625263604", "store": "app_store", "activated_at":
"2020-02-10T19:14:02.000000+0000", "renewed_at": "2020-02-10T19:29:00.000000+0000", "unsubscribed_at": null, "billing_issue_detected_at": null, "is_in_grace_period": false,
"active_introductory_offer_type": null, "active_promotional_offer_type": null, "active_promotional_offer_id": null, "cancellation_reason": null, "is_sandbox": true },
"basic_subscription_unlimited": { "is_active": true, "is_lifetime": false, "expires_at": "2021-02-27T11:00:30.000000+0000", "starts_at": null, "will_renew": false,
"vendor_product_id": "basic_subscription_unlimited", "base_plan_id": "basic_prepaid", "vendor_transaction_id": "1000000632277988", "vendor_original_transaction_id":
"1000000632277988", "store": "app_store", "activated_at": "2020-02-27T11:00:30.000000+0000", "renewed_at": null, "unsubscribed_at": null, "billing_issue_detected_at": null,
"is_in_grace_period": false, "active_introductory_offer_type": null, "active_promotional_offer_type": null, "active_promotional_offer_id": null, "cancellation_reason": null,
"is_sandbox": true } }, "non_subscriptions": null } }
```

Learn more about responses in the API Objects section**.**

Revoke subscription from a user

```
text title="Text" POST: /profiles/{profile_id_or_customer_user_id}/paid-access-levels/{access_level}/revoke/
```

Path parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------ | :------ | :---------- | | **profile*id*orcustomer*user*id*** | str | âœ… | âŒ | Adapty profile ID or developer's internal ID | | **accesslevel** | str | âœ… | âŒ | ID (slug) of a paid access level. Find it in Adapty Dashboard |

Request parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------ | :------ | :---------- | | **is_refund** | bool | âœ… | âŒ | Whether this subscription is revoked due to a refund |

Revokes user's subscription by setting **unsubscribed_at** to current datetime, and **expires_at** to a maximum of current **starts_at** and current datetime (to avoid **expires_at** being less than **starts_at**). If there is a transaction associated with this paid access level, this transaction expiration is also set to the new **expires_at** value. If **is_refund** is **true**, the transaction is marked as a refund, and revenue is set to zero.

Validate a purchase from Stripe, provide access level to a customer, and import his transaction history from Stripe

```
POST: /api/v1/sdk/purchase/stripe/token/validate/
```

:::warning This request must use a different Content-Type: `Content-Type: application/vnd.api+json'` :::

Request parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------ | :------ | :---------- | | **customer_user_id** | str | âœ… | âŒ | Developer's internal customer ID | | **stripe_token** | str | âœ… | âŒ | Token of a Stripe object that represents a unique purchase. Could either be a token of Stripe's Subscription (`sub_XXX`) or Payment Intent (`pi_XXX`). |

Sample request:

```
json title="CURL" curl --location 'https://api.adapty.io/api/v1/sdk/purchase/stripe/token/validate/' \ --header 'Content-Type: application/vnd.api+json' \ --header 'Authorization:
Api-Key <PUBLIC_OR_PRIVATE_KEY' \ --data-raw '{ "data": { "type": "stripe_receipt_validation_result", "attributes": { "customer_user_id": "<CUSTOMER_USER_ID>", "stripe_token":
"sub_1OM8brJT1bIG45BdDRFOHWAU" } } }'
```

Validates a purchase using provided Stripe token using the credentials of Stripe in your App Settings inside Adapty Dashboard. If the purchase is valid, the transaction history is imported from Stripe to the profile in Adapty with the specified customer*user*id. If there was no profile with this customer*user*id before â€” it will be created.

Profile events are generated along the way and imported transactions are counted towards MTR.

Get info about a user

```
text title="Text" GET: /profiles/{profile_id_or_customer_user_id}/
```

Path parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------ | :------ | :---------- | | **profile*id*orcustomer*user*_id** | str | âœ… | âŒ | Adapty profile ID or developer's internal ID |

The response example is the same as for Prolong/grant a subscription for a user.

To get an extended response, add Key **"extended"** with any value to Query Params. It works only for the GET request.

| Property | Type | Required | Nullable | Description | | :---------- | :--- | :------ | :------ | :---------- | **created*at*** | ISO 8601 date | âœ… | âŒ | The date when the profile was created, usually equals the installation date | | **email** | str | âœ… | âœ… | User's email | | **phonenumber** | str | âœ… | âœ… | User's phone number | | **att*status*** | str | âœ… | âœ… | | | **firstname** | str | âœ… | âœ… | User's first name | | **last*name*** | str | âœ… | âœ… | User's last name | | **username** | str | âœ… | âœ… | Username | | **gender** | str | âœ… | âœ… | User's gender | | **birthday** | ISO 8601 date | âœ… | âœ… | User's birthday | | **idfa** | str | âœ… | âœ… | The Identifier for Advertisers, assigned by Apple to a user's device. | | **idfv** | str | âœ… | âœ… | The Identifier for Vendors (IDFV) is a

code assigned to all apps by one developer and is shared across all apps by that developer on your device. | | **advertisingid** | str | âœ… | âœ… | The Advertising ID is a unique identifier offered by the Android Operating System that advertisers might use to uniquely identify you. | | **appsflyer*id*** | str | âœ… | âœ… | An AppsFlyer ID, automatically created id by AppsFlyer for every new install of an app. | | **amplitudeuser*id*** | str | âœ… | âœ… | The Amplitude User Id property specified and OneSignal's External User Id property needs to be set for message data of that device to be tracked. | | **amplitudedevice*id*** | str | âœ… | âœ… | The Amplitude Device ID, directly comes from your users' devices. | | **mixpaneluser*id*** | str | âœ… | âœ… | User ID from Mixpanel. | | **appmetricaprofile*id*** | str | âœ… | âœ… | User profile ID from AppMetrica. | | **appmetricadevice*id*** | str | âœ… | âœ… | AppMetrica Device Id. | | **facebookanonymous_id** | str | âœ… | âœ… | Facebook Anonymous ID. |

Create a user

```
text title="Text" POST: /profiles/
```

Request parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **customer_user_id** | str | âœ… | âŒ | |

Sample request:

```
json title="Json" { "customer_user_id": "123456" }
```

The response is the same as the GET request (**extended** parameter does not work here).

You can also set the user's attributes the same way as in the PATCH method.

Set the user's attribute

```
text title="Text" PATCH: /profiles/{profile_id_or_customer_user_id}/
```

Path parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **profile*id*or*customer*user_id** | str | âœ… | âŒ | Adapty profile ID or developer's internal ID |

Request parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | ip*country* | *str* | *âŒ* | *âœ…* | *Country code in the two-format, eg. US.* | | *email* | *str* | *âŒ* | *âœ…* | | | *phone*number | str | âŒ | âœ… | | | first*name* | *str* | *âŒ* | *âœ…* | | | *last*name | str | âŒ | âœ… | | | gender | str | âŒ | âœ… | User's gender. | | birthday | date | âŒ | âœ… | Date in YYYY-MM-DD format, eg. 1990-10-31. |

If you'd like to set custom attributes, you can pass them in `custom_attributes` dictionary. A maximum of 10 custom attributes for the profile are allowed to be set. Only strings and floats are allowed as values, booleans will be converted to floats.

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **attribute_key** | str | âœ… | âŒ | Only letters, numbers, dashes, points, and underscores are allowed. The attribute key must be no more than 30 characters. | | **attribute_value** | str\|float | âœ… | âœ… | The attribute value must be no more than 30 characters. Send an empty value or null to delete the attribute. |

Sample request:

```
json title="Json" { "phone_number": "+18003330000", "custom_attributes": { "grade": 10, "favorite_topic": "sports" } }
```

The response is the same as the GET request (**extended** parameter does not work here).

Delete user's data

```
DELETE /profiles/{profile_id_or_customer_user_id}/delete
```

Path parameters:

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **profile*id*or*customer*user_id** | str | âœ… | âŒ | Adapty profile ID or developer's internal ID |

Calling this endpoint enables the deletion of a user's profile and all related data, rendering it inaccessible to the client. Any profile history linked to the deleted profile will be detached, and integration events previously sent to integrations will be deleted from the event feed.

Should another profile make a purchase from the device with the same Apple ID (or when subscription is restored), the profile history will be reassigned to the new profile, and integration events will be reissued.

Please be aware that this endpoint does not support bulk deletion, therefore each request must be handled individually. For managing a substantial number of users, it is advisable to execute requests concurrently.

---

title: "API objects" description: ""

metadataTitle: ""

Objects

Adapty API has JSON objects so you can understand a response structure and wrap it into your code.

All datetime values are ISO 8601, for example, "2020-01-15T15:10:36.517975+0000".

Profile

Info about the customer and his subscription.

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **profile_id** | UUID | âœ… | âŒ | Adapty profile ID | | **customer_user_id** | str | âœ… | âœ… | User ID in developerâ€™s (your) system. | | **paid_access_levels** | dict | âœ… | âœ… | Dictionary where the keys are paid access level identifiers configured by a developer in the Adapty Dashboard. Values are CustomerAccessLevel objects. Can be null if the customer has no access levels | | **subscriptions** | dict | âœ… | âœ… | Dictionary where the keys are vendor product IDs. Values are Subscription objects. Can be null if the customer has no subscriptions | | **non_subscriptions** | dict | âœ… | âœ… | Dictionary where the keys are vendor product ids. Values are an array of Non-Subscription objects. Can be null if the customer has no purchases. | | **custom_attributes** | dict | âœ… | âœ… | The dictionary that collects the profile's all custom attributes about its own users. A maximum of 10 custom attributes for the profile are allowed to be set. Only strings and floats are allowed as values, booleans will be converted to floats. | | **total_revenue_usd** | float | âœ… | âŒ | Float value, it is equal to all total revenue USD which earned the profile. |

CustomerAccessLevel

Info about customerâ€™s access level.

Configure which products unlock premium (paid) features in your app using the Adapty Dashboard. Then, check in the app for the access level available for a user, for example in the way below (in Swift)

```
swift title="Swift" if (profile.paidAccessLevels["premium"]?.isActive == true) { /* Grant user access to paid functions of the app */ }
```

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- | | **id** | str | âœ… | âŒ | Paid Access Level identifier configured by a developer in the Adapty Dashboard | | **is_active** | bool | âœ… | âŒ | Boolean indicating whether the paid access level is active | | **expires_at** | ISO 8601 date | âœ… | âœ… | The datetime when the access level will expire. It may be in the past and may be null for lifetime access | | **starts_at** | ISO 8601 date | âœ… | âœ… | The datetime when the access level will be active. May be in the future | | **is_lifetime** | bool | âœ… | âŒ | Boolean indicating whether the paid access level is active for a lifetime (no expiration date). If set to true you shouldnâ€™t use expires_at | | **vendor_product_id** | str | âœ… | âœ… | Identifier of the product in the vendor system (App Store/Google Play/Stripe, etc.) that unlocked this access level | | **base*plan*id** | str | âœ… | âœ… | Base plan ID in the Google Play Store or price ID in Stripe. | | **store** | str | âœ… | âœ… | Store where the product was purchased. Possible values are: **app_store**, **play_store**, and **adapty** | | **activated_at** | ISO 8601 date | âœ… | âŒ | The datetime when the access level was activated. May be in the future | | **renewed_at** | ISO 8601 date | âœ… | âœ… | The datetime when the access level was renewed | | **will_renew** | bool | âœ… | âŒ | Boolean indicating whether an auto-renewable subscription is set to renew | | **is_in_grace_period** | bool | âœ… | âŒ | Boolean indicating whether auto-renewable subscription is in the grace period | | **unsubscribed_at** | ISO 8601 date | âœ… | âœ… | The datetime when an auto-renewable subscription was cancelled. Subscription can still be active, it means that auto-renewal is turned off. It will be set to null if the user reactivates the subscription | | **billing_issue_detected_at** | ISO 8601 date | âœ… | âœ… | The datetime when a billing issue was detected and a vendor was not able to charge the card. Subscription can still be active. Will set to null if the charge is successful | | **active_introductory_offer_type** | str | âœ… | âœ… | The type of active introductory offer. Possible values are: **free_trial**, **pay_as_you_go**, and **pay_up_front**. If the value is not null it means that the offer was applied during the current subscription period | | **active_promotional_offer_type** | str | âœ… | âœ… | The type of active promotional offer. Possible values are: **free_trial**, **pay_as_you_go**, and **pay_up_front**. If the value is not null it means that the offer was applied during the current subscription period |

Subscription

Info about vendor subscription. You donâ€™t have to use this object in most cases, **CustomerPaidAccessLevel** is the preferred way to work with access to the app. When using this object, you need to implement processing logic for each subscription period (a week, a month, a year, lifetime).

| Param | Type | Required | Nullable | Description | | :---------- | :--- | :------- | :------- | :---------- |

---------- | | **is_active** | bool | âœ… | âŒ | Boolean indicating whether the subscription is active **expires_at** | ISO 8601 date | âœ… | âœ… | The datetime when access level will expire. May be in the past and may be null for lifetime access | | **starts_at** | ISO 8601 date | âœ… | âœ… | The datetime when the subscription will be active. May be in the future for season subscriptions | | **is_lifetime** | bool | âœ… | âŒ | Boolean that indicates whether the subscription is active for a lifetime without an expiration date. If set to true you shouldnâ€™t use expires_at | | **vendor_product_id** | str | âœ… | âœ… | Identifier of the product in vendor system (App Store/Google Play etc.) | | **base*plan*id** | str | âœ… | âœ… | Base plan ID in the Google Play Store or price ID in Stripe. | | **vendor_transaction_id** | str | âœ… | âœ… | Transaction id in the vendor system | | **vendor_original_transaction_id** | str | âœ… | âœ… | Original transaction id in vendor system. For auto-renewable subscription, this will be the ID of the first transaction in the subscription | | **store** | str | âœ… | âœ… | Store where the product was purchased. Possible values are: **app_store**, **play_store**, and **adapty** | | **activated_at** | ISO 8601 date | âœ… | âŒ | The datetime when the access level was activated. May be in the future | | **renewed_at** | ISO 8601 date | âœ… | âœ… | The datetime when the access level was renewed | | **will_renew** | bool | âœ… | âŒ | Boolean indicating whether an auto-renewable subscription is set to renew. If a user did not cancel a subscription, will be set to true | | **is_in_grace_period** | bool | âœ… | âŒ | Boolean indicating whether an auto-renewable subscription is in the grace period | | **unsubscribed_at** | ISO 8601 date | âœ… | âœ… | The datetime when an auto-renewable subscription was canceled. Subscription can still be active, it just means that auto-renewal is turned off. Will be set to null if the user reactivates the subscription | | **billing_issue_detected_at** | ISO 8601 date | âœ… | âœ… | The datetime when the billing issue was detected (vendor was not able to charge the card). Subscription can still be active. Will be set to null if the charge is successful. | | **active_introductory_offer_type** | str | âœ… | âœ… | The type of active introductory offer. Possible values are: **free_trial**, **pay_as_you_go**, and **pay_up_front**. If the value is not null it means that the offer was applied during the current subscription period | | **active_promotional_offer_type** | str | âœ… | âœ… | The type of active promotional offer. Possible values are: **free_trial**, **pay_as_you_go**, and **pay_up_front**. If the value is not null it means that the offer was applied during the current subscription period | | **is_sandbox** | bool | âœ… | âŒ | Boolean indicating whether the product was purchased in the sandbox or production environment |

Non Subscription

Info about non-subscription purchases. These can be one-time (consumable) products, unlocks (like new map unlock in the game), etc.

| Param | Type | Required | Nullable | Description | | :---------- | :------- | :------- | :---------- | | **purchase_id** | str | âœ… | âŒ | Identifier of the purchase in Adapty. You can use it to ensure that youâ€™ve already processed this purchase, for example tracking one-time product | | **vendor_product_id** | str | âœ… | âœ… | Identifier of the product in vendor system (App Store/Google Play etc.) | | **vendor_transaction_id** | str | âœ… | âœ… | Transaction ID in the vendor system | | **vendor_original_transaction_id** | str | âœ… | âœ… | Original transaction ID in vendor system. For auto-renewable subscription, this will be the ID of the first transaction in the subscription | | **store** | str | âœ… | âœ… | Store where the product was purchased. Possible values are **app_store**, **play_store**, **adapty** | | **purchased_at** | ISO 8601 date | âœ… | âŒ | The datetime when the product was purchased | | **is_one_time** | bool | âœ… | âŒ | Boolean indicating whether the product should only be processed once. For example, if a user purchased 500 coins in a game. If true, the purchase will be returned by Adapty API one time only | | **is_sandbox** | bool | âœ… | âŒ | Boolean indicating whether the product was purchased in a sandbox or production environment. |

---

title: "Migration guide to Adapty SDK v.3.x or later" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

Adapty SDK v.3.0 brings support for the new exciting Adapty Paywall Builder, the new version of the no-code user-friendly tool to create paywalls. With its maximum flexibility and rich design capabilities, your paywalls will become most effective and profitable.

Upgrade to new Paywall Builder consists of:

1. Upgrade to Adapty SDK v3.x via Swift Package Manager or via CocoaPods. Please note that the AdaptyUI library is deprecated and is now included as part of AdaptySDK.
2. Migration of your Paywall Builder paywalls to new Paywall Builder.

Reinstall Adapty SDK v3.x via Swift Package Manager

1. Delete AdaptyUI SDK package dependency from your project, you won't need it anymore.
2. Even though you have it already, you'll need to re-add the Adapty SDK dependency. For this, in Xcode, open **File** -> **Add Package Dependency...**. Please note the way to add package dependencies can differ in XCode versions. Refer to XCode documentation if necessary.
3. Enter the repository URL `https://github.com/adaptyteam/AdaptySDK-iOS.git`
4. Choose the version, and click the **Add package** button.
5. Choose the modules you need:
    1. **Adapty** is the mandatory module
    2. **AdaptyUI** is an optional module you need if you plan to use the Adapty Paywall Builder.
6. Xcode will add the package dependency to your project, and you can import it. For this, in the **Choose Package Products** window, click the **Add package** button once again. The package will appear in the **Packages** list.

Reinstall Adapty SDK v3.x via CocoaPods

1. Add Adapty to your `Podfile`. Choose the modules you need:

    1. **Adapty** is the mandatory module.
    2. **AdaptyUI** is an optional module you need if you plan to use the Adapty Paywall Builder.

2. `shell title="Podfile" pod 'Adapty', '~> 3.0.1' pod 'AdaptyUI', '~> 3.0.1' # optional module needed only for Paywall Builder`

3. Run:

    ```
    sh title="Shell" pod install
    ```

This creates a `.xcworkspace` file for your app. Use this file for all future development of your application.

Activate Adapty and AdaptyUI SDK modules. Before v3.0, you did not activate AdaptyUI, remember to **add AdaptyUI activation**. Parameters are not changes, so keep them as is.

```swift // In your AppDelegate class: import Adapty import AdaptyUI // Only if you are going to use AdaptyUI

let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) .with(customerUserId: "YOUR*USER*ID") .with(idfaCollectionDisabled: false) .with(ipAddressCollectionDisabled: false)

Adapty.activate(with: configurationBuilder) { error in // handle the error }

// Only if you are going to use AdaptyUI AdaptyUI.activate() `</TabItem> <TabItem value="kotlin" label="SwiftUI" default>` swift title="" import Adapty import AdaptyUI // Only if you are going to use AdaptyUI

@main struct SampleApp: App { init() let configurationBuilder = Adapty.Configuration .Builder(withAPIKey: "PUBLIC*SDK*KEY") .with(observerMode: false) // optional .with(customerUserId: "YOUR*USER*ID") // optional .with(idfaCollectionDisabled: false) // optional .with(ipAddressCollectionDisabled: false) // optional

```
    Adapty.activate(with: configurationBuilder) { error in
      // handle the error
    }

        // Only if you are going to use AdaptyUI
            AdaptyUI.activate()
}

var body: some Scene {
    WindowGroup {
        ContentView()
    }
}
```

} ```

## Migrate your paywalls to new Paywall Builder

Starting with Adapty SDK v3.x, legacy Paywall Builder paywalls are no longer supported. Migrate your existing legacy Paywall Builder paywalls to the new Paywall Builder, one at a time. During this migration, Adapty will create a new version of each paywall that is compatible with the updated Paywall Builder. The old version, compatible with the legacy Paywall Builder, will remain unchanged, so users with app versions using Adapty SDK v2.x or earlier will still see their paywalls as before.

Paywalls designed without using a Paywall Builder are not affected.

---

title: "What's new in Adapty SDK 2.6" description: ""

## metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

> ðŸ’¡ If you are migrating from Adapty SDK 1.x.x, we recommend you to read What's new in Adapty SDK 2.0 article first.
```

Android SDK

:::note Don't forget to configure offers in Google Play Console in Adapty

Google Billing Library v5 and v6 changed the way offers work. Be sure to follow Google Play offers guide to configure them properly. :::

With the release of Adapty SDK version 2.6.1, we are excited to announce support for . This update includes several improvements and additions to the public API for the native Android, Flutter, React Native. Unity SDK also supports the new versions of billing library but starting with 2.7.0.

Here are the key changes and enhancements:

1. **Unified subscription details:** We have removed separate properties for free trials on Android to provide a more streamlined experience.* *Instead, we have introduced an optional* `subscriptionDetails` *property that consolidates all subscription-related properties.* *It includes an `introductoryOfferPhases` property, which is a list that can contain up to two phases: the free trial phase and the introductory price phase.

   | | Before (Kotlin) | After (Kotlin) | | :---------- | :---------- | :---------- | | Checking discount offer eligibility | `product.introductoryOfferEligibility == AdaptyEligibility.ELIGIBLE` | `product.subscriptionDetails?.introductoryOfferEligibility == AdaptyEligibility.ELIGIBLE` | | Introductory price info | `product.introductoryDiscount` | `product.subscriptionDetails?.introductoryOfferPhases?.firstOrNull { it.paymentMode in listOf(PaymentMode.PAY_UPFRONT, PaymentMode.PAY_AS_YOU_GO) }` | | Free trial info | `product.freeTrialPeriod` | `product.subscriptionDetails?.introductoryOfferPhases?.firstOrNull { it.paymentMode == PaymentMode.FREE_TRIAL }?.subscriptionPeriod` | | Free trial info (localized) | `product.localizedFreeTrialPeriod` | `product.subscriptionDetails?.introductoryOfferPhases?.firstOrNull { it.paymentMode == PaymentMode.FREE_TRIAL }?.localizedSubscriptionPeriod` |

2. **Payment mode for discounts:** To align with iOS, we have added the `paymentMode` property to the `AdaptyProductDiscountPhase` entity.
3. **Renewal type for subscriptions:** We have introduced the `renewalType` property in the `AdaptyProductSubscriptionDetails` entity to accommodate the two types of subscriptions available on Google Play: auto-renewable and prepaid.
4. **Price entity updates:** The `price`, `localizedPrice`, `currencyCode` and `currencySymbol` properties have been moved from `AdaptyPaywallProduct` to a new entity called `Price`.
5. **SKU details update:** The `skuDetails` property in `AdaptyPaywallProduct` has been renamed to `productDetails` to reflect the use of original product entities from Google.
6. **Eligibility status update:** In `AdaptyEligibility`, we have replaced the `UNKNOWN`value with `NOT_APPLICABLE`. The latter is used for products that cannot contain offers, such as `prepaid` products in the Google Play console.
7. **Personalized offers**: We have added a boolean parameter `isOfferPersonalized` to `makePurchase()`, with a default value of false. For more information, refer to the following documentation.
8. **Offer identification:** The `offerId` property has been added to the `AccessLevel` and `Subscription` entities in `AdaptyProfile`. It is an optional field that represents the discount offer ID from Google Play. Additionally, we want to draw your attention to the fact that the `vendorProductId` in these entities may contain either `productId` only or `productId:basePlanId`.
9. **Replacement mode:** We have renamed `ProrationMode` to `ReplacementMode`, and the constants have been adjusted to align with Google's standards.

For more detailed information and step-by-step guides on adding products and base plans to the Google Play Store, refer to our documentation.

We hope these updates enhance your experience with Adapty SDK and the integration with Google's new billing system.

Cross-platform SDKs migration

Determine introductory offer eligibility

Adapty SDK 2.4.x and older:

```javascript // Adapty 2.4.x and older

try { final products = await Adapty().getPaywallProducts(paywall: paywall); final product = products[0]; // don't forget to check products count before accessing

if (product.introductoryOfferEligibility == AdaptyEligibility.eligible) { // display offer } else if (product.introductoryOfferEligibility == AdaptyEligibility.ineligible) { // user is not eligible for this offer } else { // Adapty SDK wasn't able to determine eligibility at this step // Refetch products with .waitForReceiptValidation policy: final products = await adapty.getPaywallProducts( paywall: paywall, fetchPolicy: AdaptyIOSProductsFetchPolicy.waitForReceiptValidation, );

// if there wasn't error, elegibility should be eligible or ineligible.

} } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } </TabItem> <TabItem value="Unity" label="Unity" default> csharp // Adapty 2.4.x and older

Adapty.GetPaywallProducts(paywall, (products, error) => { var product = products[0]; // don't forget to check products count and error before accessing

if (product.IntroductoryOfferEligibility == Adapty.Eligibility.Eligible) { // display offer } else if (product.IntroductoryOfferEligibility == Adapty.Eligibility.Ineligible) { // user is not eligible for this offer } else { // Adapty SDK wasn't able to determine eligibility at this step // Refetch products with .WaitForReceiptValidation policy: Adapty.GetPaywallProducts(paywall, IOSProductsFetchPolicy.WaitForReceiptValidation, (products, error) => { // if there wasn't error, elegibility should be eligible or ineligible. }); } }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript // Adapty 2.4.x and older

import {adapty,OfferEligibility} from 'react-native-adapty';

const products = await adapty.getPaywallProducts(paywall); const product = products[0]; // or any other product

switch (product.introductoryOfferEligibility) { case OfferEligibility.Eligible: // display offer case OfferEligibility.Ineligible: // user is not eligible for this offer case OfferEligibility.Unknown: // Adapty SDK wasn't able to determine eligibility at this step // Refetch products with 'waitForReceiptValidation' policy:
} ```

#### Adapty SDK 2.6.0 and newer:

```javascript // Adapty 2.6.0+

try { final products = await Adapty().getPaywallProducts(paywall: paywall); // at this step you can display products // but you shouldn't display offers as eligibilities are not determined yet

final eligibilities = await Adapty().getProductsIntroductoryOfferEligibility(products: products); final introEligibility = eligibilities["your*product*id"];

switch (introEligibility) { case AdaptyEligibility.eligible: // display offer break; default: // don't display offer break; } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } </TabItem> <TabItem value="Unity" label="Unity" default> csharp // Adapty 2.6.0+

Adapty.GetPaywallProducts(paywall, (products, error) => { // at this step you can display products
// but you shouldn't display offers as eligibilities are not determined yet

Adapty.GetProductsIntroductoryOfferEligibility(products, (eligibilities, error) => { var introEligibility = eligibilities["your*product*id"]; switch (introEligibility) { case Eligibility.Eligible: // display offer break; default: // don't display offer break; } }); }); </TabItem> <TabItem value="RN" label="React Native (TS)" default> typescript // Adapty 2.6.0+

import {adapty,OfferEligibility} from 'react-native-adapty';

const products = await adapty.getPaywallProducts(paywall); const eligibilityMap = await adapty.getProductsIntroductoryOfferEligibility(products);

const introEligibility = eligibilityMap["your*product*id"];

if (intoEligibility === OfferEligibility.Eligible) { // display offer return; } // user is not eligible ```

### Displaying product and offers

#### Adapty SDK 2.4.x and older:

```javascript // Adapty 2.4.x and older

try { final products = await Adapty().getPaywallProducts(paywall: paywall); final product = products[0];

final titleString = product.localizedTitle; final priceString = product.localizedPrice;

// Introductory Offer final introductoryDiscount = product.introductoryDiscount;

if (introductoryDiscount != null) { final introPrice = introductoryDiscount.localizedPrice; final introPeriod = introductoryDiscount.localizedSubscriptionPeriod; final introNumberOfPeriods = introductoryDiscount.localizedNumberOfPeriods; }

// Free Trial for Android final freeTrialPeriod = product.freeTrialPeriod; final localizedFreeTrialPeriod = product.localizedFreeTrialPeriod;

// Promo Offer for iOSÂ¡ final promoOfferId = product.promotionalOfferId; final promoDisount = product.discounts.firstWhere((element) => element.identifier == promoOfferId);
```

if (promoDisount != null) { final promoPrice = promoDisount.localizedPrice; final promoPeriod = promoDisount.localizedSubscriptionPeriod; final promoNumberOfPeriods = promoDisount.localizedNumberOfPeriods; } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp // Adapty 2.4.0+

Adapty.GetPaywallProducts(paywall, (products, error) => { // Do not forget to check arrays lenghts and nullable properties!

var product = products[0];

var titleString = product.LocalizedTitle; var priceString = product.LocalizedPrice;

// Introductory Offer var introductoryDiscount = product.IntroductoryDiscount;

if (introductoryDiscount != null) { var introPrice = introductoryDiscount.LocalizedPrice; var introPeriod = introductoryDiscount.LocalizedSubscriptionPeriod; var introNumberOfPeriods = introductoryDiscount.LocalizedNumberOfPeriods; }

// Free Trial for Android var freeTrialPeriod = product.AndroidFreeTrialPeriod; var localizedFreeTrialPeriod = product.AndroidLocalizedFreeTrialPeriod;

// Promo Offer for iOS var promoOfferId = product.PromotionalOfferId; ProductDiscount promoDisount = null;

foreach (var discount in product.Discounts) { if (discount.Identifier == promoOfferId) { promoDisount = discount; break; } }

if (promoDisount != null) { var promoPrice = promoDisount.LocalizedPrice; var promoPeriod = promoDisount.LocalizedSubscriptionPeriod; var promoNumberOfPeriods = promoDisount.LocalizedNumberOfPeriods; } }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript // Adapty 2.4.x and older

import {adapty} from 'adapty';

const paywall = await adapty.getPaywall('MY_PAYWALL'); const products = await adapty.getPaywallProducts(paywall); const product = products[0]; // or any filter

const title = product.localizedTitle; const price = product.localizedPrice;

// Introductory offer const introDiscount = product.introductoryDiscount; if (introDiscount) { const introPrice = introDiscount.localizedPrice; const introPeriod = introDiscount.localizedSubscriptionPeriod; const introNumberOfPeriods = introDiscount.localizedNumberOfPeriods; }

// Free Trial for Android const freeTrialPeriod = product.android?.freeTrialPeriod; const localizedFreeTrialPeriod = product.android?.localizedFreeTrialPeriod;

// iOS Promo Offer const promoOfferId = product.ios?.promotionalOfferId; const promoDiscount = product.ios?.discounts.find(discount => discount.ios?.identifier === promoOfferId,); if (promoDiscount) { const promoPrice = promoDiscount.localizedPrice; const promoPeriod = promoDiscount.localizedSubscriptionPeriod; const promoNumberOfPeriods = promoDiscount.localizedNumberOfPeriods; }

```

#### Adapty SDK 2.6.0 and newer:

```javascript // Adapty 2.6.0+

try { final products = await Adapty().getPaywallProducts(paywall: paywall); final product = products[0];

final titleString = product.localizedTitle; final priceString = product.localizedPrice;

// It is possible to have more than one introductory offer (e.g. on Android) final introductoryOffer = product.subscriptionDetails?.introductoryOffer.first;

if (introductoryOffer != null) { final introPrice = introductoryOffer.price.localizedString; final introPeriod = introductoryOffer.localizedSubscriptionPeriod; final introNumberOfPeriods = introductoryOffer.localizedNumberOfPeriods; }

// Promo Offer final promotionalOfferEligibility = product.subscriptionDetails?.promotionalOfferEligibility ?? false; final promotionalOffer = product.subscriptionDetails?.promotionalOffer;

if (promotionalOfferEligibility && promotionalOffer != null) { final promoPrice = promotionalOffer.price.localizedString; final promoPeriod = promotionalOffer.localizedSubscriptionPeriod; final promoNumberOfPeriods = promotionalOffer.localizedNumberOfPeriods; } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { // handle the error } `</TabItem> <TabItem value="Unity" label="Unity" default>` csharp // Adapty 2.6.0+

Adapty.GetPaywallProducts(paywall, (products, error) => { // Do not forget to check arrays lenghts and nullable properties!

var product = products[0];

var titleString = product.LocalizedTitle; var priceString = product.Price.LocalizedString;

// It is possible to have more than one introductory offer (e.g. on Android) var introductoryOffer = product.SubscriptionDetails.IntroductoryOffer[0];

if (introductoryOffer != null) { var introPrice = introductoryOffer.Price.LocalizedString; var introPeriod = introductoryOffer.LocalizedSubscriptionPeriod; var introNumberOfPeriods = introductoryOffer.LocalizedNumberOfPeriods; }

// Promo Offer var promotionalOfferEligibility = product.SubscriptionDetails.PromotionalOfferEligibility; var promotionalOffer = product.SubscriptionDetails.PromotionalOffer;

if (promotionalOfferEligibility && promotionalOffer != null) { var promoPrice = promotionalOffer.Price.LocalizedString; var promoPeriod = promotionalOffer.LocalizedSubscriptionPeriod; var promoNumberOfPeriods = promotionalOffer.LocalizedNumberOfPeriods; } }); `</TabItem> <TabItem value="RN" label="React Native (TS)" default>` typescript // Adapty 2.6.0+

import {adapty} from 'react-native-adapty';

const paywall = await adapty.getPaywall('MY_PAYWALL'); const products = await adapty.getPaywallProducts(paywall); const product = products[0]; // or any filter

const title = product?.localizedTitle; const price = product?.localizedDescription;

// It is possible to have more than one introductory offer (e.g. on Android) const introductoryOffer = product?.subscriptionDetails?.introductoryOffers?.find( (offer, index) => index === 0, ); if (introductoryOffer) { const introPrice = introductoryOffer.price.localizedString; const introPeriod = introductoryOffer.localizedSubscriptionPeriod; const introNumberOfPeriods = introductoryOffer.localizedNumberOfPeriods; }

// Promo Offer const promotionalOffer = product?.subscriptionDetails?.ios?.promotionalOffer;

if (promotionalOffer) { const promoPrice = promotionalOffer.price.localizedString; const promoPeriod = promotionalOffer.localizedSubscriptionPeriod; const promoNumberOfPeriods = promotionalOffer.localizedNumberOfPeriods; }

```

---

title: "What's new in Adapty SDK 2.0" description: ""

metadataTitle: ""

Explore what's new in the latest version of our SDK, and learn how to plan the migration process.

- iOS
- Android
- Flutter
- React Native
- Unity

---

title: "iOS â€“ What's new" description: ""

metadataTitle: ""

In the new version of the Adapty SDK, we've made quite a lot of changes to the internal implementation of our SDK, applying all of our accumulated experience. We also redesigned our public API and relationships between some entities so that it causes as little misunderstanding as possible and reduces the number of errors made by developers.

First, let's look at the main things that have been changed, and then let's discuss each item individually.

- Profile â€“ we have renamed `PurchaserInfo` to `AdaptyProfile`. This also affected all related functions.
- Paywalls â€“ now you can get only the necessary paywall.
- Products â€“ we separated the concept of paywall and product, so now the developer first asks for the paywall they want, and then the corresponding product array.
- Fallback Paywalls â€“ `.setFallbackPaywalls` method now accepts `Data` type as a parameter.
- Promotional offers â€“ we removed the `offerId` parameter from `.makePurchase` method.
- Introductory offer eligibility â€“ instead of true/false we give an extended list of options.
```

- Products Fetch Policy â€“ we have added the ability to explicitly get products after we send the receipt to our servers.
- Logging â€“ We redesigned the logging module so that you can integrate your systems into it and use the logs as you see fit.
- Swift â€“ our iOS SDK is now written in pure Swift.

We've also compiled a full list of changes, which can be found on the release page on our GitHub

Profile

Before:
Previously, all information that related to the user was in the `PurchaserInfo` model. This information was obtained by calling the `.getPurchaserInfo(forceUpdate:)` method:

```
swift title="Swift" // AdaptySDK 1.x.x Adapty.getPurchaserInfo(forceUpdate: Bool) { (purchaserInfo, error) in if error == nil { // check the accesss } }
```

After:
Now we have renamed the model to `AdaptyProfile`. Also it affected the corresponding method:

```
swift title="Swift" // AdaptySDK 2.0.0 Adapty.getProfile { result in if let profile = try? result.get() { // check the access } }
```

Motivation:
This name reflects the essence of the model much more correctly, because not every user is a subscriber.

This change also affected the delegate method `.didReceiveUpdatedProfile` to `.didLoadLatestProfile`

Paywalls

Before:
Previously, developers used to query an array of paywalls and then search that array for the desired element.

```
swift title="Swift" // AdaptySDK 1.x.x Adapty.getPaywalls(forceUpdate: Bool) { (paywalls, products, error) in if error == nil { // retrieve the products from paywalls } }
```

After:
We have significantly simplified this use case, so now you can get only the requested object, without touching the rest.

```
swift title="Swift" // AdaptySDK 2.0.0 Adapty.getPaywall("YOUR_PLACEMENT_ID") { result in switch result { case let .success(paywall): // the requested paywall case let
.failure(error): // handle the error } }
```

Motivation:
In addition to simplifying the most common usage scenario, we also significantly reduce the load on our servers, which will allow us to get a response from the server as quickly as possible.

Products

Before:
Previously the product entity was a part of the paywall, so you could use it right after `.getPaywalls` method was done. Also you could use products out of the paywalls context.

```
swift title="Swift" // AdaptySDK 1.x.x Adapty.getPaywalls(forceUpdate: Bool) { (paywalls, products, error) in if error == nil { // retrieve the products from paywalls } }
```

After:
Once you have obtained the desired paywall, you can query the products array for it. Now the product entity is independent, although it can only exist in the context of the paywall.

```
swift title="Swift" // AdaptySDK 2.0.0 Adapty.getPaywallProducts(paywall: paywall) { result in switch result { case let .success(products): // the requested products array case let
.failure(error): // handle the error } }
```

Motivation:
We believe that this architecture will provide more flexibility in terms of receiving paywalls and products (for example, now you are not blocked by Apple when you receive a paywall), and will also optimize the load on the servers, which will speed up the response. Also, this approach is less error-prone.

:::warning Products outside the paywalls

If you for some reason want to work with a product (or an array of products), please create a paywall for it. This approach is great for scaling and analytics. :::

Fallback paywalls

Before:
We used to accept fallback paywalls in `String` format:

```
swift title="Swift" if let path = Bundle.main.path(forResource: "fallback_paywalls", ofType: "json"), let paywalls = try? String(contentsOfFile: path, encoding: .utf8) {
Adapty.setFallbackPaywalls(paywalls) }
```

After:
We have now replaced this function with one that takes a parameter of type `Data`:

```
swift title="Swift" if let urlPath = Bundle.main.url(forResource: "fallback_paywalls", withExtension: "json"), let paywallsData = try? Data(contentsOf: urlPath) {
Adapty.setFallbackPaywalls(paywallsData) }
```

Motivation:
We changed the type of accepted parameter to avoid unnecessary conversions.

Promotional offers

If your paywall has an active promotional offer for the product you are attempting to purchase, Adapty will automatically apply that offer at the time of purchase.

Before:
The `makePurchase` function required the `offerId` parameter to be passed explicitly:

```
swift title="Swift" Adapty.makePurchase(product: <product>, offerId: <offerId>) { (purchaserInfo, receipt, appleValidationResult, product, error) in if error == nil { // successful
purchase } }
```

After:
Starting with version 2.0 this parameter is no longer present. The Adapty SDK automatically uses the `promotionalOfferId` field to apply the discount, taking the value of the `promotionalOfferEligibility` field into account.

```
swift title="Swift" Adapty.makePurchase(paywall: paywall, product: product) { result in switch result { case let .success(response): // successful purchase case let .failure(error):
// handle the error } }
```

:::warning Adapty signs the request according to Apple guidelines, please make sure you've uploaded Subscription Key in Adapty Dashboard when using promotional offers. :::

Introductory offer eligibility

The `Product` entity has the `introductoryOfferEligibility` property, it determines whether the introductory offer is available to the user (for example, a free trial period) or has already been used.

Before:
`introductoryOfferEligibility` was a Boolean value

After:
`introductoryOfferEligibility` is an enumeration

```
swift title="Swift" // AdaptySDK 2.0.0 public enum AdaptyEligibility { case unknown case ineligible case eligible }
```

Motivation:
StoreKit does not provide a convenient and reliable way to determine this value, so we have to do it by analyzing the receipt from the system. Since there are cases when this receipt is missing, we decided to inform you about these situations using the value `unknown`. We recommend working with `unknown` in the same way as `ineligible`.

Products fetch policy

As mentioned in the previous section, StoreKit version 1 does not allow you to reliably determine the value of the `introductoryOfferEligibility` without analyzing the receipt. Despite the fact that a missing receipt at startup is a pretty rare situation, we have added the ability to explicitly get products after we send the receipt to our servers.

Before:
In previous versions, the `syncTransactionsHistory` function was added for such situations, which often conflicted with internal SDK calls, slowing it down.

```
swift title="Swift" Adapty.syncTransactionsHistory { params, products, error in Adapty.getPaywall("paywall_id") { paywall, error in // use paywall and products } }
```

After:
Now this mechanism is implemented much more reliably, we will try to request a receipt in its unavailability in advance, and there is a special parameter of `.getPaywallProducts` function to get products with a correct `introductoryOfferEligibility`:

```
swift title="Swift" Adapty.getPaywallProducts(paywall: paywall, fetchPolicy: .waitForReceiptValidation) { result in if let products = try? result.get() { // update your UI } }
```

Motivation:
The call sequence for the correct functioning of the system was unobvious and less reliable.
We recommend first requesting products without overriding `fetchPolicy`, and then immediately rendering the UI. If you get back objects with an unknown `introductoryOfferEligibility` value, you can re-request products with `.waitForReceiptValidation` policy and update the UI afterward.
Read more about handling such a scenario in the *Displaying Paywalls & Products* section.

Logging

It is critical for a library such as ours to have an understandable and customizable logging system. Just as before, our SDK has several levels of logging, but now you can also override the stream the messages will be sent to. By default, messages will be sent to the console, but you can optionally write them to a file or send them to your favorite logging system.

```
swift title="Swift" Adapty.setLogHandler { level, message in logToSomeFancyLoggingSystem("Adapty \(level): \(message)") }
```

iOS and Swift

Despite the fact that technically the first version of the SDK was written in Swift, we were forced to maintain backward compatibility with Objective-C, as there are a huge number of projects created using this obsolete programming language.
This approach prevented us from using Swift to its full potential. We did not take full advantage of the Value Types, as well as the many other features available only in Swift.

For example, we can now happily use the built-in Result type. This led to a slight change in callbacks in many places in the public API.

Before:
The callback contains a lot of variables, including an error.

```
swift title="Swift" Adapty.makePurchase(product: <product>, offerId: <offerId>) { (purchaserInfo, receipt, appleValidationResult, product, error) in if error == nil { // successful purchase } }
```

After:
A much more elegant and familiar way to handle the result

```
swift title="Swift" Adapty.makePurchase(paywall: paywall, product: product) { result in switch result { case let .success(response): // successful purchase case let .failure(error): // handle the error } }
```

In lieu of a conclusion

In this article, we have listed the most significant changes introduced in the new version, which can be seen in the public API. However, most of the improvements are hidden "under the hood" and are not mentioned here. Of course, we've completely updated our documentation to reflect the new release, so you can feel free to use it.

You can find the complete list of changes on the release page.

Stay tuned for more updates!

---

title: "Android â€“ What's new" description: ""

metadataTitle: ""

In the new version of the Adapty SDK, we've made quite a lot of changes to the internal implementation of our SDK, applying all of our accumulated experience. We also redesigned our public API and relationships between some entities so that it causes as little misunderstanding as possible and reduces the number of errors made by developers.

First, let's look at the main things that have been changed, and then let's discuss each item individually.

- Profile â€“ we have renamed `PurchaserInfo` to `AdaptyProfile`. This also affected all related functions.
- Paywalls â€“ now you can get only the necessary paywall.
- Products â€“ we separated the concept of paywall and product, so now the developer first asks for the paywall they want, and then the corresponding product list.

We've also compiled a full list of changes, which can be found on the release page on our GitHub

Profile

Before:
Previously, all information that related to the user was in the `PurchaserInfo` model. This information was obtained by calling the `.getPurchaserInfo(forceUpdate)` method:

```
kotlin title="Kotlin" // AdaptySDK 1.x.x Adapty.getPurchaserInfo(forceUpdate) { purchaserInfo, error -> if (error == null) { // check the access } }
```

After:
Now we have renamed the model to `AdaptyProfile`. Also it affected the corresponding method:

```
kotlin title="Kotlin" // AdaptySDK 2.0.0 Adapty.getProfile { result -> if (result is AdaptyResult.Success) { val profile = result.value //check the access } }
```

Motivation:
This name reflects the essence of the model much more correctly, because not every user is a subscriber.

This change also affected the method `.setOnPurchaserInfoUpdatedListener()` to `.setOnProfileUpdatedListener()`

Paywalls

Before:
Previously, developers used to query a list of paywalls and then search that list for the desired element.

```
kotlin title="Kotlin" // AdaptySDK 1.x.x Adapty.getPaywalls(forceUpdate) { paywalls, products, error -> if (error == null) { // retrieve the products from paywalls } }
```

After:
We have significantly simplified this use case, so now you can get only the requested object, without touching the rest.

```
kotlin title="Kotlin" // AdaptySDK 2.0.0 Adapty.getPaywall("YOUR_PLACEMENT_ID") { result -> when (result) { is AdaptyResult.Success -> { val paywall = result.value // the requested paywall } is AdaptyResult.Error -> { val error = result.error // handle the error } } }
```

Motivation:
In addition to simplifying the most common usage scenario, we also significantly reduce the load on our servers, which will allow us to get a response from the server as quickly as possible.

Products

Before:
Previously the product entity was a part of the paywall, so you could use it right after `.getPaywalls` method was done. Also you could use products out of the paywalls context.

```
kotlin title="Kotlin" // AdaptySDK 1.x.x Adapty.getPaywalls(forceUpdate) { paywalls, products, error -> if (error == null) { // retrieve the products from paywalls } }
```

After:
Once you have obtained the desired paywall, you can query the list of products for it. Now the product entity is independent, although it can only exist in the context of the paywall.

```
kotlin title="Kotlin" // AdaptySDK 2.0.0 Adapty.getPaywallProducts(paywall) { result -> when (result) { is AdaptyResult.Success -> { val products = result.value // the requested products } is AdaptyResult.Error -> { val error = result.error // handle the error } } }
```

Motivation:
We believe that this architecture will provide more flexibility in terms of receiving paywalls and products (for example, now you are not blocked by Google when you receive a paywall), and will also optimize the load on the servers, which will speed up the response. Also, this approach is less error-prone.

:::warning Products outside the paywalls

If you for some reason want to work with a product (or an array of products), please create a paywall for it. This approach is great for scaling and analytics. :::

In lieu of a conclusion

In this article, we have listed the most significant changes introduced in the new version, which can be seen in the public API. However, most of the improvements are hidden "under the hood" and are not mentioned here. Of course, we've completely updated our documentation to reflect the new release, so you can feel free to use it.

You can find the complete list of changes on the release page.

Stay tuned for more updates!

---

title: "Flutter â€“ What's new" description: ""

metadataTitle: ""

In the new version of the Adapty SDK, we've made quite a lot of changes to the internal implementation of our SDK, applying all of our accumulated experience. We also redesigned our public API and relationships between some entities so that it causes as little misunderstanding as possible and reduces the number of errors made by developers.

First, let's look at the main things that have been changed, and then let's discuss each item individually.

- Profile â€“ we have renamed `AdaptyPurchaserInfo` to `AdaptyProfile`. This also affected all related functions.
- Paywalls â€“ now you can get only the necessary paywall.
- Products â€“ we separated the concept of paywall and product, so now the developer first asks for the paywall they want, and then the corresponding product array.
- Promotional offers â€“ we removed the `offerId` parameter from `.makePurchase` method.
- Introductory offer eligibility â€“ instead of true/false we give an extended list of options.
- Products Fetch Policy â€“ we have added the ability to explicitly get products after we send the receipt to our servers.

We've also compiled a full list of changes, which can be found on the release page on our GitHub

Profile

Before:
Previously, all information that related to the user was in the `PurchaserInfo` model. This information was obtained by calling the `.getPurchaserInfo(forceUpdate:)` method:

```
javascript title="Flutter" // AdaptySDK 1.x.x try { AdaptyPurchaserInfo purchaserInfo = await Adapty().getPurchaserInfo({bool forceUpdate = false}); // "premium" is an identifier of default access level if (purchaserInfo.accessLevels['premium']?.isActive ?? false) { // grant access to premium features } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

After:
Now we have renamed the model to `AdaptyProfile`. Also, it affected the corresponding method:

```
javascript title="Flutter" // AdaptySDK 2.0.0 try { AdaptyProfile profile = await Adapty().getProfile(); // "premium" is an identifier of default access level if (profile.accessLevels['premium']?.isActive ?? false) { // grant access to premium features } } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

Motivation:
This name reflects the essence of the model much more correctly because not every user is a subscriber.

This change also affected the corresponding stream: `didReceivePurchaserInfoStream` was renamed to `didUpdateProfileStream`

Paywalls

Before:
Previously, developers used to query an array of paywalls and then search that array for the desired element.

```
javascript title="Flutter" // AdaptySDK 1.x.x try { final GetPaywallsResult getPaywallsResult = await Adapty().getPaywalls(forceUpdate: Bool); final List<AdaptyPaywall> paywalls = getPaywallsResult.paywalls; } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

After:
We have significantly simplified this use case, so now you can get only the requested object, without touching the rest.

```
javascript title="Flutter" // AdaptySDK 2.0.0 try { final AdaptyPaywall paywall = await Adapty().getPaywall(id: 'YOUR_PLACEMENT_ID'); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

Motivation:
In addition to simplifying the most common usage scenario, we also significantly reduce the load on our servers, which will allow us to get a response from the server as quickly as possible.

Products

Before:
Previously the product entity was a part of the paywall so that you could use it right after `.getPaywalls` method was done. Also, you could use products out of the paywall context.

```
javascript title="Flutter" // AdaptySDK 1.x.x try { final GetPaywallsResult getPaywallsResult = await Adapty().getPaywalls(forceUpdate: Bool); final List<AdaptyPaywall> paywalls = getPaywallsResult.paywalls; // retrieve the products from paywalls } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

After:
Once you have obtained the desired paywall, you can query the products array for it. Now the product entity is independent, although it can only exist in the context of the paywall.

```
javascript title="Flutter" // AdaptySDK 2.0.0 try { final products = await Adapty().getPaywallProducts(paywall: paywall, fetchPolicy: fetchPolicy); // the requested products array } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

Motivation:
We believe that this architecture will provide more flexibility in terms of receiving paywalls and products (for example, now you are not blocked by Apple when you receive a paywall), and will also optimize the load on the servers, which will speed up the response. Also, this approach is less error-prone.

:::warning Products outside the paywalls

If you for some reason want to work with a product (or an array of products), please create a paywall for it. This approach is great for scaling and analytics. :::

Promotional offers

If your paywall has an active promotional offer for the product you are attempting to purchase, Adapty will automatically apply that offer at the time of purchase.

Before:
The `makePurchase` function required the `offerId` parameter to be passed explicitly:

```
javascript title="Flutter" // AdaptySDK 1.x.x try { final MakePurchaseResult makePurchaseResult = await Adapty().makePurchase(<product>, offerId: <offer_id>); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

After:
Starting with version 2.0 this parameter is no longer present. The Adapty SDK automatically uses the `promotionalOfferId` field to apply the discount, taking the value of the `promotionalOfferEligibility` field into account.

```
javascript title="Flutter" // AdaptySDK 2.0.0 try { final profile = await Adapty().makePurchase(product: product); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { onUnknownErrorOccured?.call(e); }
```

:::warning Adapty signs the request according to Apple guidelines, please make sure you've uploaded Subscription Key in Adapty Dashboard when using promotional offers. :::

Introductory offer eligibility

The `Product` entity has the `introductoryOfferEligibility` property, it determines whether the introductory offer is available to the user (for example, a free trial period) or has already been used.

Before:
`introductoryOfferEligibility` was a Boolean value

After:
`introductoryOfferEligibility` is an enumeration

```
javascript title="Flutter" // AdaptySDK 2.0.0 enum AdaptyEligibility { unknown, ineligible, eligible, }
```

Motivation:
StoreKit does not provide a convenient and reliable way to determine this value, so we have to do it by analyzing the receipt from the system. Since there are cases when this receipt is missing, we decided to inform you about these situations using the value `unknown`. We recommend working with `unknown` in the same way as `ineligible`.

Products fetch policy

As mentioned in the previous section, StoreKit version 1 does not allow you to reliably determine the value of the `introductoryOfferEligibility` without analyzing the receipt. Despite the fact that a missing receipt at startup is a pretty rare situation, we have added the ability to explicitly get products after we send the receipt to our servers.

This mechanism is implemented in this way: we will try to request a receipt in its unavailability in advance, and there is a special parameter of `.getPaywallProducts` function to get products with a correct `introductoryOfferEligibility`:

```
javascript title="Flutter" // AdaptySDK 2.0.0 try { final products = await Adapty().getPaywallProducts(paywall: paywall, fetchPolicy:
AdaptyIOSProductsFetchPolicy.waitForReceiptValidation); } on AdaptyError catch (adaptyError) { // handle the error } catch (e) { }
```

Motivation:
The call sequence for the correct functioning of the system was unobvious and less reliable.
We recommend first requesting products without overriding `fetchPolicy`, and then immediately rendering the UI. If you get back objects with an unknown `introductoryOfferEligibility` value, you can re-request products with `.waitForReceiptValidation` policy and update the UI afterward.
Read more about handling such a scenario in the *Displaying Paywalls & Products* section.

In lieu of a conclusion

In this article, we have listed the most significant changes introduced in the new version, which can be seen in the public API. However, most of the improvements are hidden "under the hood" and are not mentioned here. Of course, we've completely updated our documentation to reflect the new release, so you can feel free to use it.

You can find the complete list of changes on the release page.

Stay tuned for more updates!

---

title: "React Native â€" What's new" description: ""

metadataTitle: ""

import Tabs from '@theme/Tabs'; import TabItem from '@theme/TabItem';

In the new version of the Adapty SDK, we've made quite a lot of changes to the internal implementation of our SDK, applying all of our accumulated experience. We also redesigned our public API and relationships between some entities so that it causes as little misunderstanding as possible and reduces the number of errors made by developers.

First, let's outline the things that have changed, and then let's discuss every item individually.

Synopsis

- Activation â€" function moved to `adapty` namespace, arguments redesigned
- Logging and debugging â€" Clear errors messages, customization
- Profile â€" Methods renamed
- Getting paywalls â€" Getting only requested paywall instead of all. `AdaptyPaywall` interface has changed
- Getting products â€" Getting list of products of a provided paywall separately. `AdaptyProduct` interface has changed
- Introductory offer eligibility â€“ Instead of a boolean, now there is an extended list of options
- Products fetch policy â€“ Ability to explicitly get products after we send the receipt to our servers
- Making purchases â€" Method renamed. Removed the `offerId` parameter
- Updating attribution â€" Arguments changed order
- Promos â€" Promo API discontinued. All methods removed
- Event listeners â€" Events changed

Activation

Methods
In SDK v1, there was a separate `activateAdapty` function, that you would import. It read one object argument with all the parameters for initialization.

In SDK v2 there are several changes:

1. It is now a method `activate` of `adapty` namespace, so you would only need to import `adapty` instance
2. All the parameters besides `sdkKey` are now expected from the second argument (refer to example)
3. For a `logLevel` parameter, you may now import JavaScript-friendly `LogLevel` to make sure you provide a valid value. TypeScript string validation remains

Example
Note, that you can switch tabs:

- `v2.0.0 (New)` is an example of basic and precise activations for SDK v2
- `v1.x.x (Previous)` is an example of basic and precise activations for a latest `v1.x.x` version

```typescript // AdaptySDK 2.0.0

import { adapty, LogLevel } from 'react-native-adapty';

// Basic setup: await adapty.activate('MY*API*KEY');

// Or precise setup: await adapty.activate('MY*API*KEY', { customerUserId: 'MY*USER*ID', observerMode: true, logLevel: LogLevel.VERBOSE, // â† can be replaced with a string 'verbose' too }); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

import { activateAdapty } from 'react-native-adapty';

// Basic setup: await activateAdapty({ sdkKey: 'MY*API*KEY', });

// Or precise setup: await activateAdapty({ sdkKey: 'MY*API*KEY', customerUserId: 'MY*USER*ID', observerMode: true, logLevel: 'verbose', }); ```

**Motivation**

1. Simplifying the most common usage scenario
2. Excluding ambiguous functions outside `adapty` scope

### Logging and debugging

In SDK v2 there are several new features:

**Error prefixes**
In SDK v2 you can now prepend a string to all Adapty error logs. It is ok to call this before initialization and wherever you want.

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.prefix = "[ADAPTY]"; ```

**Understandable logs**
If you were trying to log error message, you would previously see "Unknown Adapty Error", as message was not handled by stdout. In SDK v2 logging errors are clear and concise.
This is an example: `[ADAPTY] #2002 (notActivated): The Adapty is not activated`

1. `[ADAPTY]` is a prefix, that you can manually set as stated above
2. `#2002` is an Adapty code for you to Google around
3. `notActivated` is a string representation of a code. It might give you enough info to do a fix
4. `The Adapty is not activated` is a `localizedDescription` field from SDK v1

**Error hooks**
You can now handle all the Adapty errors from any given place with `onError` hook. It will send to a callback all the registered AdaptyErrors right after they are created

```typescript title="Typescript" import { AdaptyError } from 'react-native-adapty';

AdaptyError.onError = error => { // ... }; ```

**Changing `logLevel` in a runtime**
Now you can change your `logLevel` without reinitializing the SDK.
```

```typescript title="Typescript" import { adapty, LogLevel } from 'react-native-adapty';

adapty.setLogLevel(LogLevel.WARN); // string 'warn' would also work ```

### Profile

**Methods**
Previously, in SDK v1, there were several methods:

1. `adapty.purchases.getInfo`
2. `adapty.profile.identify`
3. `adapty.profile.logout`
4. `adapty.profile.update`

In SDK v2, methods are renamed:

1. `adapty.purchases.getInfo` â†’ `adapty.getProfile`
2. `adapty.profile.identify` â†’ `adapty.identify`
3. `adapty.profile.logout` â†’ `adapty.logout`
4. `adapty.profile.update` â†’ `adapty.updateProfile`

**Example**
Note, that you can switch tabs:

- `v2.0.0 (New)` is an example of getting user profile for SDK v2
- `v1.x.x (Previous)` is an example of getting user profile for a latest `v1.x.x` SDK version

```typescript // AdaptySDK 2.0.0

await adapty.identify();

const profile = await adapty.getProfile();

await adapty.updateProfile({firstName: "John", lastName: "Doe" });

await adapty.logout(); `</TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default>` typescript // AdaptySDK 1.x.x

await adapty.profile.identify();

const profile = await adapty.purchases.getInfo({ forceUpdate: true });

await adapty.profile.update({firstName: "John", lastName: "Doe" });

await adapty.profile.logout(); ```

**Interfaces**

First of all, interfaces are renamed to improve readability and understanding. It is unlikely, that you've imported these to your project, but if you are, refer here:

1. `AdaptyPurchaserInfo` â†’ `AdaptyProfile`
2. `AdaptyProfile` â†’ `AdaptyProfileParameters`
3. `AdaptyPaidAccessLevelsInfo` â†’ `AdaptyAccessLevel`
4. `AdaptyNonSubscriptionsInfo` â†’ `AdaptyNonSubscription`
5. `AdaptySubscriptionsInfo` â†’ `AdaptySubscription`
6. `AdaptyOfferType` â†’ `OfferType`
7. `AdaptyVendorStore` â†’ `VendorStore`

If you are using JavaScript, there are several new objects that may guarantee you safe values. You may import and use them as enums:

1. `OfferType` for example in `AdaptyAccessLevel.activeIntroductoryOfferType` and many more
2. `CancellationReason` for example in `AdaptyAccessLevel.cancellationReason`
3. `VendorStore` for example in `AdaptyAccessLevel.store`
4. `AppTrackingTransparencyStatus` in `AdaptyProfileParameters.appTrackingTransparencyStatus`
5. `Gender` in `AdaptyProfileParameters.gender`

Below you may find an extensive diff for every *profile* interface:

```diff title="Profile interfaces diff" // Returned fromgetProfile, makePurchase, restorePurchases` -interface AdaptyPurchaserInfo { +interface AdaptyProfile { accessLevels?: Record; + customAttributes: Partial; - customerUserId: string; + customerUserId?: string; nonSubscriptions?: Record; }

-interface AdaptyPaidAccessLevelsInfo { +interface AdaptyAccessLevel { - activatedAt: string; + activatedAt: Date; - activeIntroductoryOfferType: AdaptyOfferType; + activeIntroductoryOfferType?: OfferType; + activePromotionalOfferId?: string; - activePromotionalOfferType: AdaptyOfferType; + activePromotionalOfferType?: OfferType; - billingIssueDetectedAt?: string; + billingIssueDetectedAt?: Date; + cancellationReason?: CancellationReason; - expiresAt: string; + expiresAt?: Date; id: string; isActive: boolean; isInGracePeriod: boolean; isLifetime: boolean; + isRefund: boolean; - renewedAt: string; + renewedAt?: Date; - startsAt: string; + startsAt?: Date; store: VendorStore; - unsubscribedAt?: string; + unsubscribedAt?: Date; + vendorOriginalTransactionId?: string; vendorProductId: string; + vendorTransactionId?: string; willRenew: boolean; }

-interface AdaptyProfile { +interface AdaptyProfileParameters { amplitudeDeviceId?: string; amplitudeUserId?: string; + analyticsDisabled?: boolean; - attStatus?: 0 | 1 | 2 | 3; + appTrackingTransparencyStatus?: AppTrackingTransparencyStatus; appmetricaDeviceId?: string; appmetricaProfileId?: string; - birthday?: Date; + birthday?: string; - customAttributes: { [key: string]: any }; + codableCustomAttributes?: { [key: string]: any }; - customerUserId: string; email?: string; facebookAnonymousId?: string; - facebookUserId: string; + firebaseAppInstanceId?: string; firstName?: string; - gender: 'male' | 'female' | 'other'; gender?: Gender; - idfa: string; lastName?: string; mixpanelUserId?: string; + oneSignalPlayerId?: string; phoneNumber?: string; + pushwooshHWID?: string; + storeCountry?: string; }

-interface AdaptyNonSubscriptionsInfo { +interface AdaptyNonSubscription { isOneTime: boolean; + isRefund: boolean; isSandbox: boolean; purchaseId: string; - purchasedAt?: string; + purchasedAt: Date; store: VendorStore; - vendorOriginalTransactionId: string; vendorProductId: String; - vendorTransactionId: string; + vendorTransactionId?: string; }

-interface AdaptySubscriptionsInfo { +interface AdaptySubscription { - activatedAt?: string; + activatedAt: Date; - activeIntroductoryOfferType: AdaptyOfferType; + activeIntroductoryOfferType?: OfferType; + activePromotionalOfferId?: string; activePromotionalOfferType?: OfferType; - billingIssueDetectedAt?: string; + billingIssueDetectedAt?: Date; - cancellationReason?: string; + cancellationReason?: CancellationReason; - expiresAt?: string; + expiresAt?: Date; isActive: boolean; isInGracePeriod: boolean; isLifetime: boolean; isRefund: boolean; isSandbox: boolean; - renewedAt?: string; + renewedAt?: Date; - startsAt?: string; + startsAt?: Date; store: VendorStore; - unsubscribedAt?: string; + unsubscribedAt?: Date; - vendorOriginalTransactionId?: string; + vendorOriginalTransactionId: string; vendorProductId: string; - vendorTransactionId?: string; + vendorTransactionId: string; willRenew: boolean; } ```

**Motivation**
This name reflects the essence of the model much more correctly, because not every user is a subscriber

### Getting paywalls

**Methods**
In SDK v1 there you could use `adapty.paywalls.getPaywalls(args?: { forceUpdate?: boolean })` that returned a list of paywalls.

In SDK v2, you can't fetch all paywalls at once. Instead you are expected to fetch the one you need via *developer ID*. Moreover, in SDK v2 fetching full products is a separate method. There are 2 methods available to you:

- `adapty.getPaywall(id: string)` fetches one paywall for a provided *developer id*
- `adapty.getPaywallProducts(paywall: AdaptyPaywall)` fetches a list of products for a provided paywall. It will be discussed in the next section

**Example**
Previously, developers had to query an array of paywalls and then search that array for the desired element. We have significantly simplified this use case, so now you can get only the requested object, without touching the rest.
Note, that you can switch tabs:

- `v2.0.0 (New)` is how you can get a paywall in a new SDK v2
- `v1.x.x (Previous)` is how you you could get a paywall and its products in SDK v1

```typescript // AdaptySDK 2.0.0

const paywall = await adapty.getPaywall('YOUR*PLACEMENT*ID'); `</TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default>` typescript // AdaptySDK 1.x.x

const { paywalls } = await adapty.paywalls.getPaywalls({ forceUpdate: true }); // Find your preferred paywall. For example: const paywall = paywalls.find(paywall => paywall.developerId === 'MY_PAYWALL'); ```

**Logging**
Previously, in SDK v1 there was a `adapty.paywalls.logShow` method to log, that user has opened a paywall.

In SDK v2 there are two separate methods now:

- `adapty.paywalls.logShow` is renamed to `adapty.logShowPaywall`
- `adapty.logShowOnboarding`

**Fallback paywalls**
Previously, in SDK v1 there was `adapty.paywalls.setFallback` method. In SDK v2 it is now called `adapty.setFallbackPaywalls`.

```typescript // AdaptySDK 2.0.0

await adapty.logShowPaywall();

await adapty.setFallbackPaywalls(jsonStr); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

await adapty.paywalls.logShow();

await adapty.paywalls.setFallback(jsonStr); ```

**Interfaces**
Below you can find all the changes introduced in v2.0.0 to `AdaptyPaywall` interface. Note, that you can switch tabs:

- `Changes` is a diff, that shows what have been removed and what have been added. Comments are provided
- `v2.0.0 (New)` is an interface representation in a new SDK v2
- `v1.x.x (Previous)` is an interface representation in a latest `v1.x.x` version

```
diff interface AdaptyPaywall { - abTestName?: string; // it is now required + abTestName: string; - customPayloadString?: string; // renamed to `remoteConfigString` + remoteConfigString?: string; + remoteConfig?: string; // parsed JSON from `remoteConfigString` - developerId: string; // renamed to `id` + id: string; - isPromo: boolean; // Promos are no longer supported name?: string; - products: AdaptyProduct[]; // Full products are no longer fetched + vendorProductIds: string[]; // List of vendor ids is fetched instead revision: number; variationId: string; - visualPaywall?: string; // Visual paywalls are not currently supported } ```typescript // AdaptySDK 2.0.0
```

export interface AdaptyPaywall { readonly abTestName: string; readonly id: string; readonly name?: string; readonly remoteConfig?: Record; readonly remoteConfigString?: string; readonly revision: number; readonly variationId: string; readonly vendorProductIds?: string[]; } </TabItem> <TabItem value="java" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

export interface AdaptyPaywall { abTestName?: string; customPayloadString?: string; developerId: string; isPromo: boolean; name?: string; products: AdaptyProduct[]; revision: number; variationId: string; visualPaywall?: string; } ```

**Motivation**

1. Simplifying the most common usage scenario
2. Reduce internet traffic, to immensely improve response time

## Getting products

**Methods**
Previously, in SDK v1 a product list was a part of `AdaptyPaywall`. Now in SDK v2 a product list is independent, although it can only exist in the context of the `AdaptyPaywall`.

There is a new method to get products: `adapty.getPaywallProducts(paywall)`.

**Example**
Note, that you can switch tabs:

- `v2.0.0 (New) is how you can query products in SDK v2
- `v1.x.x (Previous)` is how you could query products in SDK v1

```typescript // AdaptySDK 2.0.0

const paywall = await adapty.getPaywall('YOUR*PLACEMENT*ID'); const products = await adapty.getPaywallProducts(paywall); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

const { paywalls } = await adapty.paywalls.getPaywalls({ forceUpdate: true }); // Find your preferred paywall. For example: const paywall = paywalls.find(paywall => paywall.developerId === 'MY_PAYWALL'); const products = paywall.products; ```

**Interface**
Two interfaces slightly changed: `AdaptyProduct` and `AdaptyProductDiscount`. You may find diff below
Interfaces renamed:

1. `AdaptyProductSubscriptionPeriod` â†’ `AdaptySubscriptionPeriod`

```diff title="Diff" interface AdaptyProduct { - currencyCode: string; + currencyCode?: string; - currencySymbol: string; + currencySymbol?: string; introductoryDiscount?: AdaptyProductDiscount; - introductoryOfferEligibility: boolean; + introductoryOfferEligibility: OfferEligibility; localizedDescription: string; - localizedPrice: string; + localizedPrice?: string; localizedSubscriptionPeriod?: string; localizedTitle: string; - paywallABTestName?: string; + paywallABTestName: string; - paywallName?: string; + paywallName: string; price: number; subscriptionPeriod: AdaptySubscriptionPeriod; - variationId?: string; + variationId: string; vendorProductId: string;

android?: {
    freeTrialPeriod?: AdaptySubscriptionPeriod;

+ localizedFreeTrialPeriod?: string; }; ios?: { discounts: AdaptyProductDiscount[]; isFamilyShareable: boolean; - promotionalOfferEligibility: boolean; + promotionalOfferEligibility: OfferEligibility; promotionalOfferId?: string; regionCode?: string; subscriptionGroupIdentifier?: string; }; }

interface AdaptyProductDiscount { + localizedNumberOfPeriods?: string; - localizedPrice: string; + localizedPrice?: string; - localizedSubscriptionPeriod: string; + localizedSubscriptionPeriod?: string; numberOfPeriods: number; price: number; subscriptionPeriod: AdaptySubscriptionPeriod;

ios?: {
    identifier?: string;
    paymentMode: OfferType;

- localizedNumberOfPeriods?: string; // now crossplatform }; } ```

**Motivation:**
We believe that this architecture will provide more flexibility in terms of receiving paywalls and products (for example, now you are not blocked by Apple when you receive a paywall), and will also optimize the load on the servers, which will speed up the response. Also, this approach is less error-prone.

:::warning Products outside the paywalls

If you for some reason want to work with a product (or an array of products), please create a paywall for it. This approach is great for scaling and analytics. :::

## Introductory offer eligibility

The `AdaptyProduct` entity has the `introductoryOfferEligibility` property, that determines whether the introductory offer is available to the user (for example, a free trial period).

In SDK v1 it was a boolean value. In SDK v2 now it is a string union `'eligible' | 'ineligible' | 'unknown''`. You can also import `OfferEligibility` enum object if you need.

Note, that you can switch tabs:

- `v2.0.0 (New)` is how you can handle `introductoryOfferEligibility` in SDK v2
- `v1.x.x (Previous)` is how you handled `introductoryOfferEligibility` in SDK v1

```typescript // AdaptySDK 2.0.0

import { OfferEligibility } from 'react-native-adapty'; // ... switch (product.introductoryOfferEligibility) { case OfferEligibility.Eligible: // or 'eligible' string // ... case OfferEligibility.Ineligible: // or 'ineligible' string // ... case OfferEligibility.Unknown: // or 'unknown' string // ... } </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

// ... if (product.introductoryOfferEligibility) { // ... } else { // ... } ```

**Motivation**
**StoreKit** does not provide a convenient and reliable way to determine this value, so we have to do it by analyzing the receipt from the system. Since there are cases when this receipt is missing, we decided to inform you about these situations using the value `unknown`. We recommend working with `unknown` in the same way as `ineligible`.

## Products fetch policy

Previously, SDK v1 did not allow you to reliably determine the value of the `introductoryOfferEligibility` without analyzing the receipt. Despite the fact that a missing receipt at startup is a pretty rare situation, we

have added the ability to explicitly get products after we send the receipt to our servers.

In SDK v2 we will try to request a receipt in its unavailability in advance, and there is a special parameter of `getPaywallProducts` function to get products with a correct `introductoryOfferEligibility`

On JavaScript you can import `FetchPolicy` object to validate the passing values.

```
typescript title="Typescript" // AdaptySDK v2.0.0 adapty.getPaywallProducts({ios: { fetchPolicy: 'waitForReceiptValidation' }});
```

Motivation
We recommend first requesting products without overriding `fetchPolicy`, and then immediately rendering the UI. If you get back objects with an unknown `introductoryOfferEligibility` value, you can re-request products with `waitForReceiptValidation` policy and update the UI afterward.
Read more about handling such a scenario in the *Displaying Paywalls & Products* section.

Making purchases

Methods
Previously, in SDK v1 there was one method: `adapty.purchases.makePurchase`, that accepted product and platform-specific offer IDs.

In SDK v2 the method is renamed to `adapty.makePurchase`. It only accepts a product now. If your paywall has an active promotional offer for the product you are attempting to purchase, Adapty will automatically apply that offer at the time of purchase.

Example
Note that you can switch tabs:

- `v2.0.0 (New)` is how you can make a purchase in SDK v2
- `v1.x.x (Previous)` is how you could make a purchase in SDK v1

```typescript // AdaptySDK v2.0.0

await adapty.makePurchase(product); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK v1.x.x

await adapty.purchases.makePurchase(product, { ios: { offerId: offerId }}); ```

:::warning Adapty signs the request according to Apple guidelines, please make sure you've uploaded Subscription Key in Adapty Dashboard when using promotional offers. :::

**Other purchase methods**

Previously, in SDK v1 there were 2 more methods:

1. `adapty.purchases.setVariationId` to inform Adapty about paywall purchases in *Observer Mode*
2. `adapty.purchases.restore`

In SDK v2 these methods have been renamed:

1. `adapty.purchases.setVariationId` â†' `adapty.setVariationId`
2. `adapty.purchases.restore` â†' `adapty.restorePurchases`

**Example**
Note that you can switch tabs:

- `v2.0.0 (New)` is how you can set a `variationId` in SDK v2
- `v1.x.x (Previous)` is how you could set a `variationId` in SDK v1

```typescript // AdaptySDK v2.0.0

await adapty.setVariationId(variationId, transactionId);

await adapty.restorePurchases(); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK v1.x.x

await adapty.purchases.setVariationId(variationId, transactionId);

await adapty.purchases.restore(); ```

## Updating attribution

**Methods**
Previously, in SDK v1 you needed to pass three arguments to an `updateAttribution` call: `networkUserId: string`, `attribution: Object`, `source: string union` consecutively.
It caused a problem for several sources: they do not provide an exposed `networkUserId`, which was handled in SDK v2.

In SDK v2, there are several changes:

1. Arguments now have different order to make `networkUserId` optional: `attribution`, `source`, then optional `networkUserId`
2. For a `source` parameter, you may now import JavaScript-friendly `AttributionSource` object to make sure you provide a valid value. TypeScript string validation remains

**Example**
Note, that you can switch tabs:

- `v2.0.0 (New)` is an example of updating attribution for SDK v2
- `v1.x.x (Previous)` is an example of updating attribution for a latest `v1.x.x` SDK version

```typescript // AdaptySDK 2.0.0

import { adapty, AttributionSource } from 'react-native-adapty';

// AppsFlyer example appsFlyer.onInstallConversionData((installData) => { const networkUserId = appsFlyer.getAppsFlyerUID();

await adapty.updateAttribution(
    installData,
    AttributionSource.AppsFlyer, // â† can be replaced with a string 'AppsFlyer' too
    networkUserId,
);

}); </TabItem> <TabItem value="kotlin" label="v1.x.x (Previous)" default> typescript // AdaptySDK 1.x.x

import { adapty } from 'react-native-adapty';

// AppsFlyer example appsFlyer.onInstallConversionData((installData) => { const networkUserId = appsFlyer.getAppsFlyerUID();

await adapty.updateAttribution(
    networkUserId,
    installData,
    'AppsFlyer',
);

}); ```

**Motivation**
As stated, with introduction of new sources, `networkUserId` became optional. Major library update allows to change order of arguments to avoid passing something like`""`

## Promos

Adapty no longer supports Promo Push API. All methods were removed, except iOS native one.

**`presentCodeRedemptionSheet` renamed**
`adapty.promo.presentCodeRedemptionSheet` â†' `adapty.presentCodeRedemptionSheet`

## Event listeners

Previously, in SDK v1 there were 3 event listeners:

- `onInfoUpdate (profile: AdaptyProfile)`
```

- `onDeferredPurchase (product: AdaptyProduct)`
- `onPromoReceived (promo: AdaptyPromo)`

In SDK v2, there are only two event listeners:

- `onLatestProfileLoad (profile: AdaptyProfile)`
- `onDeferredPurchase (profile: AdaptyProfile)`

Deferred purchase event now sends a `AdaptyProfile` callback instead of a product. `onLatestProfileLoad` replaces `onInfoUpdate` and works exactly at the same times as previously

In lieu of a conclusion

In this article, we have listed the most significant changes introduced in the new version, which can be seen in the public API. However, most of the improvements are hidden "under the hood" and are not mentioned here. Of course, we've completely updated our documentation to reflect the new release, so you can feel free to use it.

You can find the complete list of changes on the release page.

Stay tuned for more updates!

---

title: "Unity â€“ What's new" description: ""

metadataTitle: ""

In the new version of the Adapty SDK, we've made quite a lot of changes to the internal implementation of our SDK, applying all of our accumulated experience. We also redesigned our public API and relationships between some entities so that it causes as little misunderstanding as possible and reduces the number of errors made by developers.

First, let's look at the main things that have been changed, and then let's discuss each item individually.

- Profile â€“ we have renamed `PurchaserInfoModel` to `Profile`. This also affected all related functions.
- Paywalls â€“ now you can get only the necessary paywall.
- Products â€“ we separated the concept of paywall and product, so now the developer first asks for the paywall they want, and then the corresponding product array.
- MakePurchase Signature â€“ we removed the `offerId` parameter from `.MakePurchase` method.
- Introductory offer eligibility â€“ instead of true/false we give an extended list of options.
- Products Fetch Policy â€“ we have added the ability to explicitly get products after we send the receipt to our servers.

We've also compiled a full list of changes, which can be found on the release page on our GitHub

Profile

Before:
Previously, all information that related to the user was in the `PurchaserInfo` model. This information was obtained by calling the `.getPurchaserInfo(forceUpdate:)` method:

```
csharp title="C#" // AdaptySDK 1.x.x Adapty.GetPurchaserInfo(forceUpdate, (purchaserInfo, error) => { if (error == null) { // check the access } });
```

After:
Now we have renamed the model to `Adapty.Profile`. Also it affected the corresponding method:

```
csharp title="C#" // AdaptySDK 2.0.0 Adapty.GetProfile((profile, error) => { if (error == null) { // check the access } });
```

Motivation:
This name reflects the essence of the model much more correctly, because not every user is a subscriber.

This change also affected the `AdaptyEventListener` method: `.OnReceiveUpdatedPurchaserInfo` was renamed to `.OnLoadLatestProfile`

Paywalls

Before:
Previously, developers used to query an array of paywalls and then search that array for the desired element.

```csharp title="C#" // AdaptySDK 1.x.x Adapty.GetPaywalls(forceUpdate, (result, error) => { if(error != null) { // handle the error return; }

var paywalls = result.Paywalls; var products = result.Products; }); ```

**After:**
We have significantly simplified this use case, so now you can get only the requested object, without touching the rest.

```csharp title="C#" // AdaptySDK 2.0.0 Adapty.GetPaywall("YOUR*PLACEMENT*ID", (paywall, error) => { if(error != null) { // handle the error return; }

// paywall - the resulting object }); ```

**Motivation:**
In addition to simplifying the most common usage scenario, we also significantly reduce the load on our servers, which will allow us to get a response from the server as quickly as possible.

### Products

**Before:**
Previously the product entity was a part of the paywall, so you could use it right after `.getPaywalls` method was done. Also you could use products out of the paywalls context.

```csharp title="C#" // AdaptySDK 1.x.x Adapty.GetPaywalls(forceUpdate, (result, error) => { if(error != null) { // handle the error return; }

var paywalls = result.Paywalls; var products = result.Products; }); ```

**After:**
Once you have obtained the desired paywall, you can query the products array for it. Now the product entity is independent, although it can only exist in the context of the paywall.

```csharp title="C#" // AdaptySDK 2.0.0 Adapty.GetPaywallProducts(paywall, (products, error) => { if(error != null) { // handle the error return; }

// products - the requested products array }); ```

**Motivation:**
We believe that this architecture will provide more flexibility in terms of receiving paywalls and products (for example, now you are not blocked by Apple when you receive a paywall), and will also optimize the load on the servers, which will speed up the response. Also, this approach is less error-prone.

:::warning Products outside the paywalls

If you for some reason want to work with a product (or an array of products), please create a paywall for it. This approach is great for scaling and analytics. :::

### MakePurchase Signature

In the previous version of sdk, you had to pass `productId`, `variationId` and `offerId` as parameters in order to make a purchase. This approach caused a lot of questions and often led to errors. In the new version you only need to pass the product object, all the necessary information sdk will get from it. If your paywall has an active promotional offer for the product you are attempting to purchase, Adapty will automatically apply that offer at the time of purchase.

**Before:**
The `.MakePurchase` function required the`productId`, `variationId` and `offerId` parameters to be passed explicitly:

```csharp title="C#" // AdaptySDK 1.x.x Adapty.MakePurchase(productId, variationId, offerId, (result, error) => { if(error != null) { // handle the error return; }

var product = result.Product; var purchaserInfo = result.PurchaserInfo; var receipt = result.Receipt; }); ```

**After:**
Starting with version 2.0 you only need to pass the `product` parameter (and `subscriptionUpdate` for Android if needed).
The Adapty SDK automatically reads `variationId` parameter and uses the `promotionalOfferId` field to apply the discount, taking the value of the `promotionalOfferEligibility` field into account.

```csharp title="C#" // AdaptySDK 2.0.0 Adapty.MakePurchase(product, (profile, error) => { if(error != null) { // handle the error return; }

// successful purchase }); ```

:::warning Adapty signs the request according to Apple guidelines, please make sure you've uploaded Subscription Key in Adapty Dashboard when using promotional offers. :::

### Introductory offer eligibility

The `PaywallProduct` entity has the `IntroductoryOfferEligibility` property, it determines whether the introductory offer is available to the user (for example, a free trial period) or has already been used.

**Before:**
`IntroductoryOfferEligibility` was a Boolean value

**After:**
`introductoryOfferEligibility` is an enumeration

```
csharp title="C#" // AdaptySDK 2.0.0 public enum Eligibility { Unknown, Ineligible, Eligible }
```

Motivation:
StoreKit does not provide a convenient and reliable way to determine this value, so we have to do it by analyzing the receipt from the system. Since there are cases when this receipt is missing, we decided to inform you about these situations using the value `unknown`. We recommend working with `unknown` in the same way as `ineligible`.

Products fetch policy

As mentioned in the previous section, StoreKit version 1 does not allow you to reliably determine the value of the `IntroductoryOfferEligibility` without analyzing the receipt. Despite the fact that a missing receipt at startup is a pretty rare situation, we have added the ability to explicitly get products after we send the receipt to our servers.

This mechanism is implemented in this way: we will try to request a receipt in its unavailability in advance, and there is a special parameter of `.GetPaywallProducts` function to get products with a correct `IntroductoryOfferEligibility`:

```csharp title="C#" // AdaptySDK 2.0.0 Adapty.GetPaywallProducts(paywall, IOSProductsFetchPolicy.WaitForReceiptValidation, (products, error) => { if(error != null) { // handle the error return; }

// update your UI }); ```
```

Motivation:
The call sequence for the correct functioning of the system was unobvious and less reliable.
We recommend first requesting products without overriding `fetchPolicy`, and then immediately rendering the UI. If you get back objects with an unknown `IntroductoryOfferEligibility` value, you can re-request products with `.WaitForReceiptValidation` policy and update the UI afterward.
Read more about handling such a scenario in the *Displaying Paywalls & Products* section.

In lieu of a conclusion

In this article, we have listed the most significant changes introduced in the new version, which can be seen in the public API. However, most of the improvements are hidden "under the hood" and are not mentioned here. Of course, we've completely updated our documentation to reflect the new release, so you can feel free to use it.

You can find the complete list of changes on the release page.

Stay tuned for more updates!