



Firebase Cloud Messaging

Using Firebase Cloud Messaging you can notify an app client that a new email or other data is available to be synced. You can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4000 bytes to an app client.

Check the [official page](#) for more information.

Setup

Before starting to use any Firebase extensions, you are required to follow some initial configuration steps. However if you've already done these for any of the other modules you can skip this configuration section and go straight to using the API functions.

- [Create Project](#)
- [Platform Setup](#) (iOS requires some additional steps)

Functions

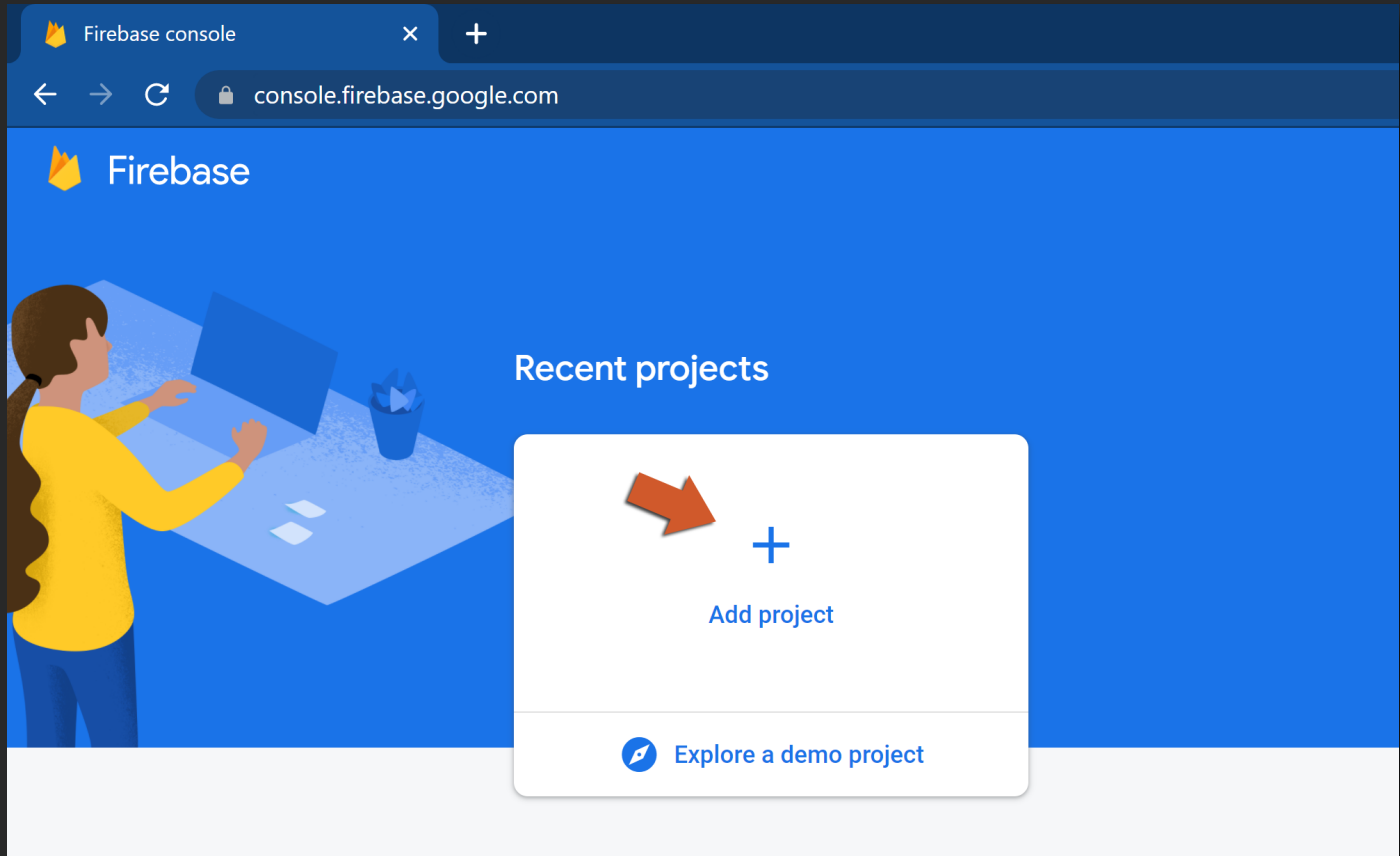
The following functions are provided for working with the Firebase Cloud Messaging extension:

- [FirebaseCloudMessaging_DeleteToken](#)
- [FirebaseCloudMessaging_GetToken](#)
- [FirebaseCloudMessaging_IsAutoInitEnabled](#)
- [FirebaseCloudMessaging_SetAutoInitEnabled](#)
- [FirebaseCloudMessaging_SubscribeToTopic](#)

Create Project

Before working with any Firebase functions, you must set up your Firebase project:

1. Go to the **Firebase Console** web site.
2. Click on **Add Project** to create your new project.



3. Enter a name for your project and click on the **Continue** button.

Let's start with a name for your project [?]

Enter your project name

my-awesome-project-id

Select parent resource


Continue


4. On the next page, make sure that **Enable Google Analytics for this project** is enabled and then click the **Continue** button:


Google Analytics for your Firebase project


Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, Predictions, and Cloud Functions.


Google Analytics enables:


 A/B testing [?]

 Crash-free users [?]

 User segmentation & targeting across
Firebase products [?]

 Event-based Cloud Functions triggers [?]

 Predicting user behavior [?]

 Free unlimited reporting [?]



Enable Google Analytics for this project
Recommended

Previous





Continue


5. Select your account and click the Create project button:

× Create a project (Step 3 of 3)


Configure Google Analytics

Choose or create a Google Analytics account ?

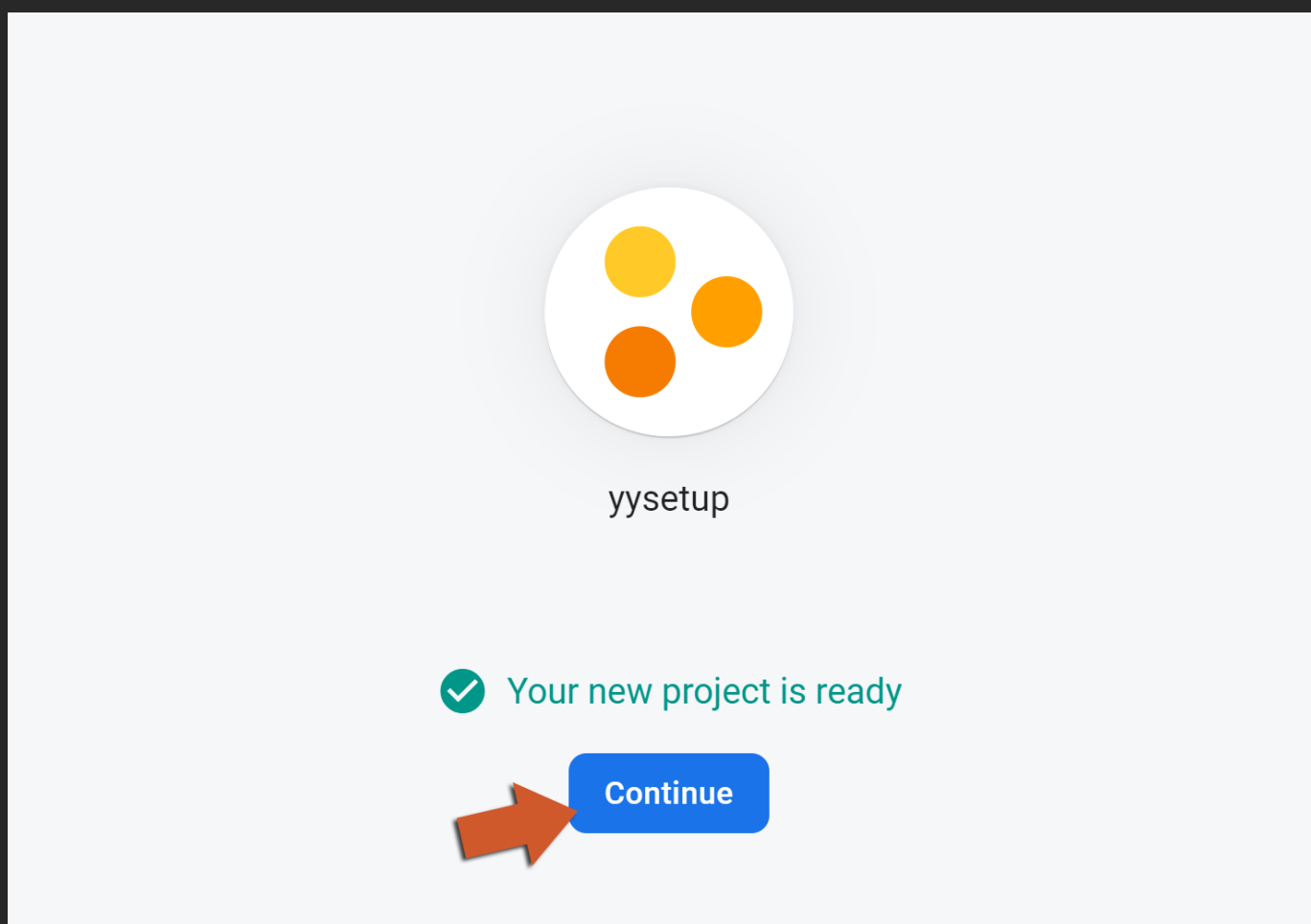
  Default Account for Firebase ▼

Automatically create a new property in this account 

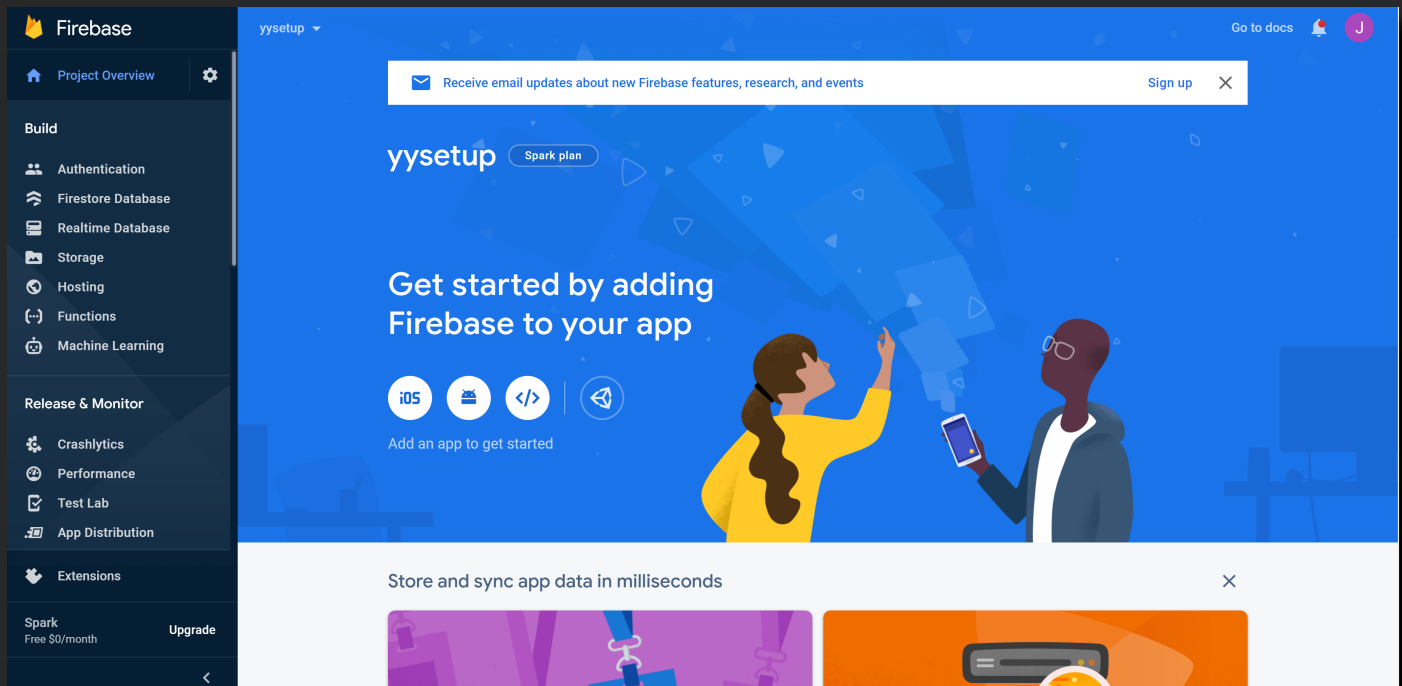
Upon project creation, a new Google Analytics property will be created in your chosen Google Analytics account and linked to your Firebase project. This link will enable data flow between the products. Data exported from your Google Analytics property into Firebase is subject to the Firebase terms of service, while Firebase data imported into Google Analytics is subject to the Google Analytics terms of service. [Learn more](#).

[Previous](#)[Create project](#)

6. Wait a moment until you project is created; after a few moments you should see the page shown below:



7. You will now be taken to your new project's home page:



8. Continue your adventure with the Firebase extensions provided for GameMaker!

Platform Setup

Firebase Cloud Messaging implementation uses SDK dependencies and therefore is only available on **Android** and **iOS** targets. In this section we will cover the required setup necessary to start using the Cloud Messaging extension on your game.

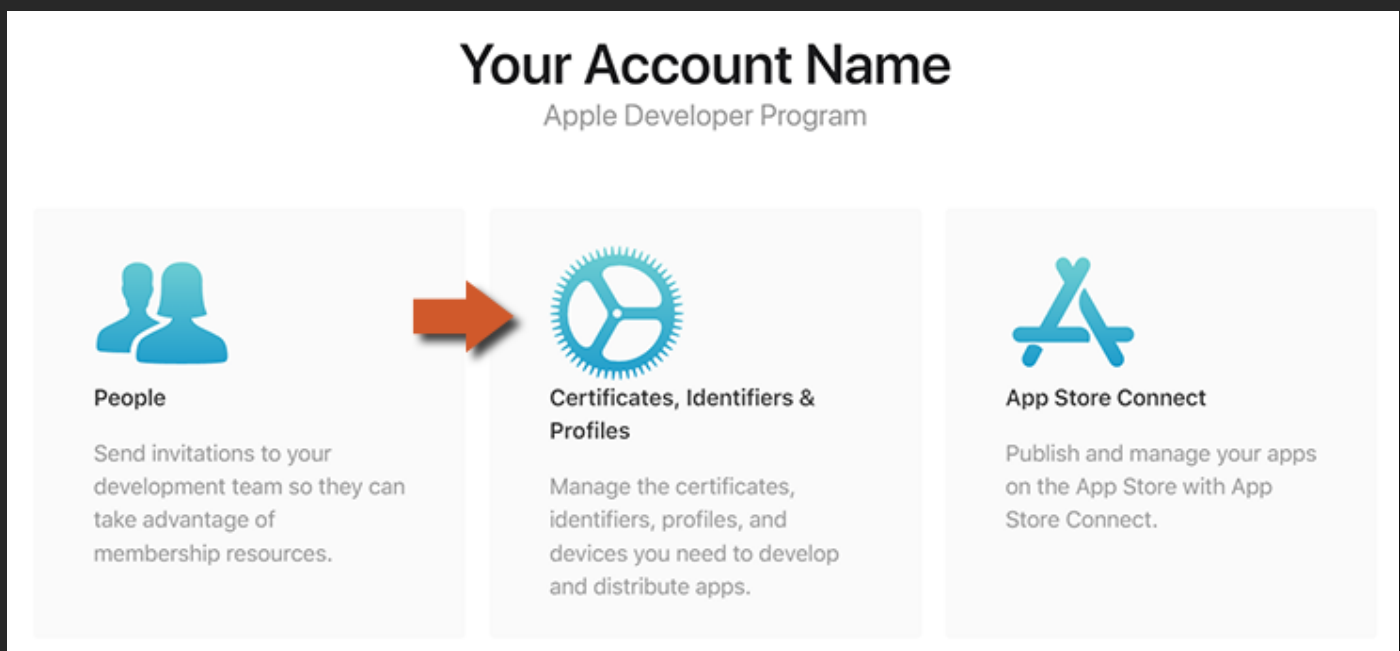
Select your target platform below and follow the simple steps to get your project up and running (you only need follow this setup once per project):

- [Android Setup](#)
- [iOS Setup](#)

Additional steps for iOS

On iOS you will need to retrieve a P8 certificate and upload it to your Firebase project to enable sending push notifications through APNs:

- Head to the [Apple Developer](#) site and select "Certificates, Identifiers & Profiles".



- Select "**Keys**" from the menu on the left, and create a new key by clicking on the plus sign.

Certificates, Identifiers & Profiles

Certificates

Keys  

Identifiers

NAME 

SERVICE

Devices

Profiles

Keys 

More

- Enter a **name** for the key, enable **Apple Push Notifications service (APNs)** and click on **Continue**.


[< All Keys](#)

Register a New Key

[Continue](#)


Key Name

You cannot use special characters such as @, &, *, ' , " , - , .

ENABLE	NAME	DESCRIPTION
<input checked="" type="checkbox"/> 	Apple Push Notifications service (APNs)	Establish connectivity between your notification server and the Apple Push Notification service. One key is used for all of your apps. Learn more

- On the next page, confirm the key details and click on **Register**.

Register a New Key

[Back](#) [Register](#) 

Key Name

Firebase APNs

ENABLE	NAME	DESCRIPTION
<input checked="" type="checkbox"/>	Apple Push Notifications service (APNs)	Establish connectivity between your notification server and the Apple Push Notification service. One key is used for all of your apps.

- Note** the information given here (key ID) and **download** the key as you will not be able to see this screen again.

View Key Details

[Download](#)[Edit](#)

Name

Firebase APNs

Key ID

0000000000

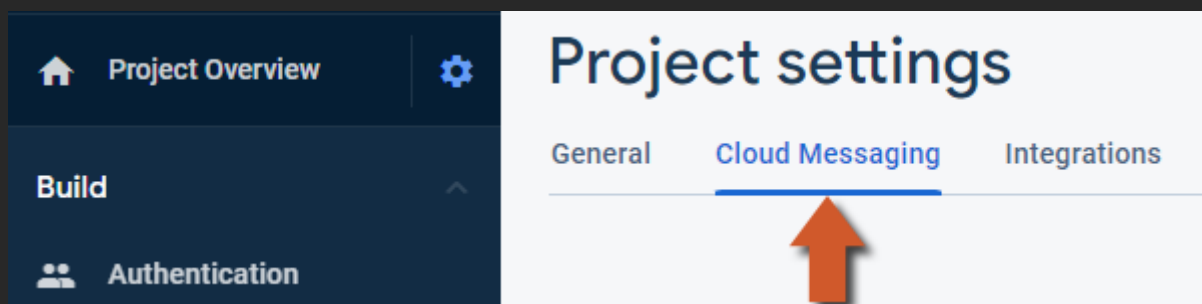
Enabled Services

NAME

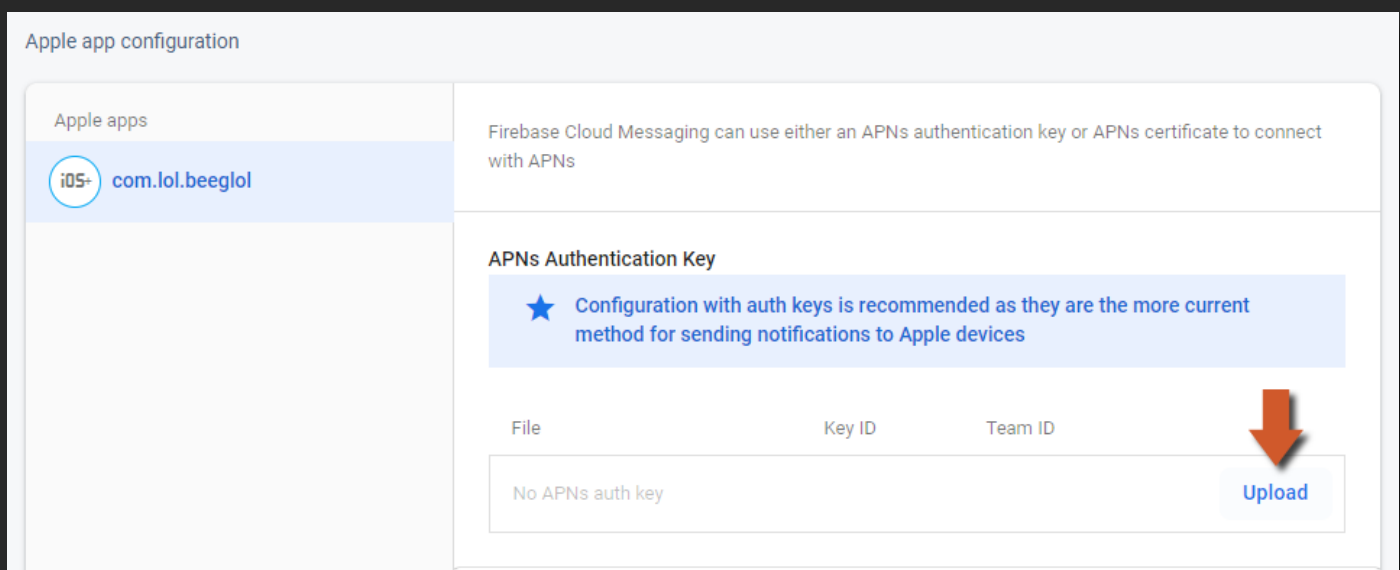
CONFIGURATION

Apple Push Notifications service (APNs)

- Go to the dashboard for your Firebase project and open the **Project Settings**. Here, open the **Cloud Messaging** tab.



- Select your iOS application, and under "APNs Authentication Key", press **Upload** to upload your key.



- Here, upload your P8 file and enter the other required details that you retrieved from the Apple Developer site.

Upload APNs auth key

APNs auth key ?

Drag file here to preview

Supports P8

Browse

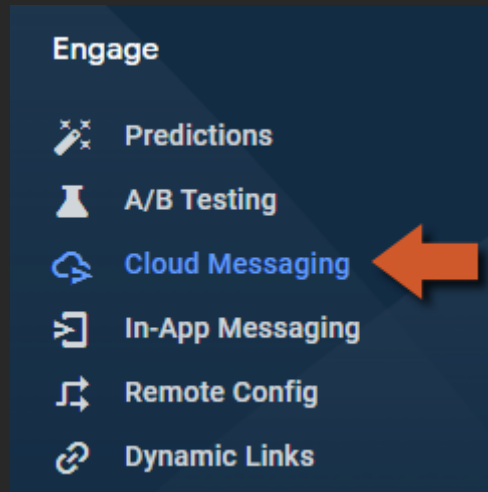
Key ID (required) ?

Enter key ID

Team ID (required) ?

Enter team ID

You can now send notifications to the iOS client game by going under "Engage" and selecting "Cloud Messaging" on your Firebase dashboard.

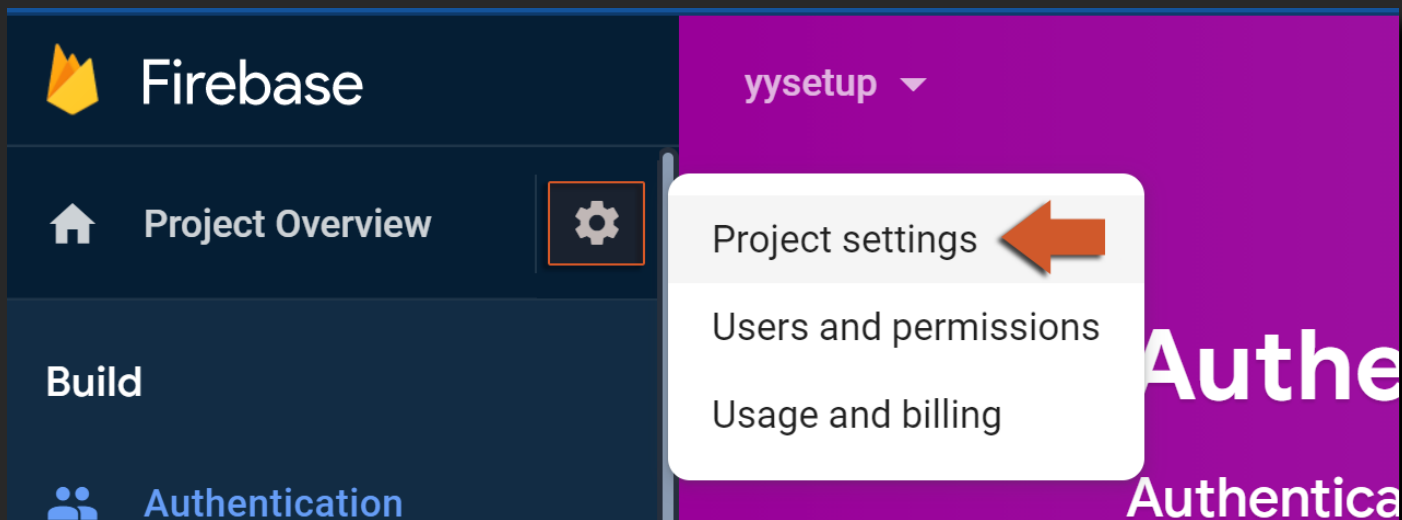


Android Setup

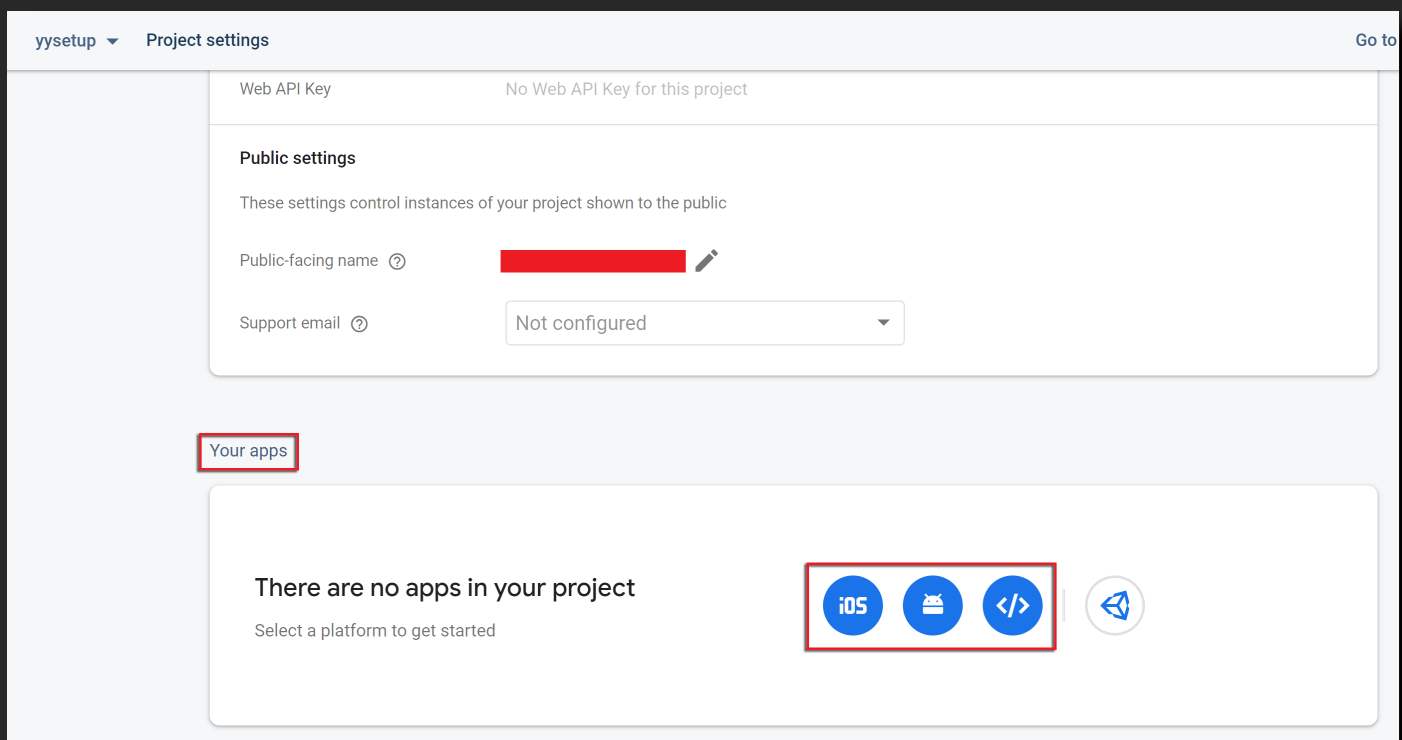
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the `google-services.json` file into your project.

IMPORTANT Please refer to [this Helpdesk article](#) for instructions on setting up an Android project.

1. Click the **Settings** icon (next to Project Overview) and select **Project settings**:



2. Now go to the **Your apps** section and click on the **Android** button:



3. On this screen you need enter your **Package name** (required), **App nickname** (optional) and **Debug signing certificate SHA-1** (required if you are using Firebase Authentication).


×

Add Firebase to your Android app

1


Register app

Android package name ?



App nickname (optional) ?

Debug signing certificate SHA-1 (optional) ?




Required for Dynamic Links, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

Register app

You can get your package name from the **Android Game Options**, and your Debug signing certificate SHA-1 from the **Android Preferences** (under Keystore):

Key Hash

Key Hash (SHA1)

 Show Key Hash

Generate Keystore

4. Click on **Download google-services.json** (make sure to save this file as we will need it in subsequent steps).

× Add Firebase to your Android app

✓ Register app

Android package name: com.yoyogames.YoyoPlayServices2, App nickname: yysetup_android

2 Download config file

Instructions for Android Studio below | [Unity](#) [C++](#)



↓ Download google-services.json

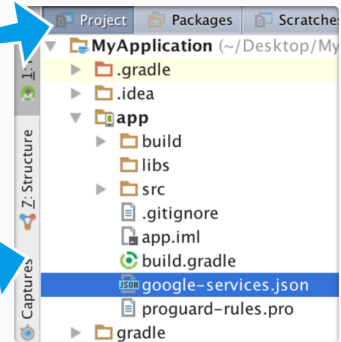
Switch to the Project view in Android Studio to see your project root directory.

Move the google-services.json file you just downloaded into your Android app module root directory.



google-services.json

Next



5. Ignore this screen, as this is already done in the extension.

× Add Firebase to your Android app

✓ Register app

Android package name: com.yoyogames.YoyoPlayServices2, App nickname: yysetup_android

✎ Download config file

3 Add Firebase SDK

Instructions for Gradle | [Unity](#) [C++](#)

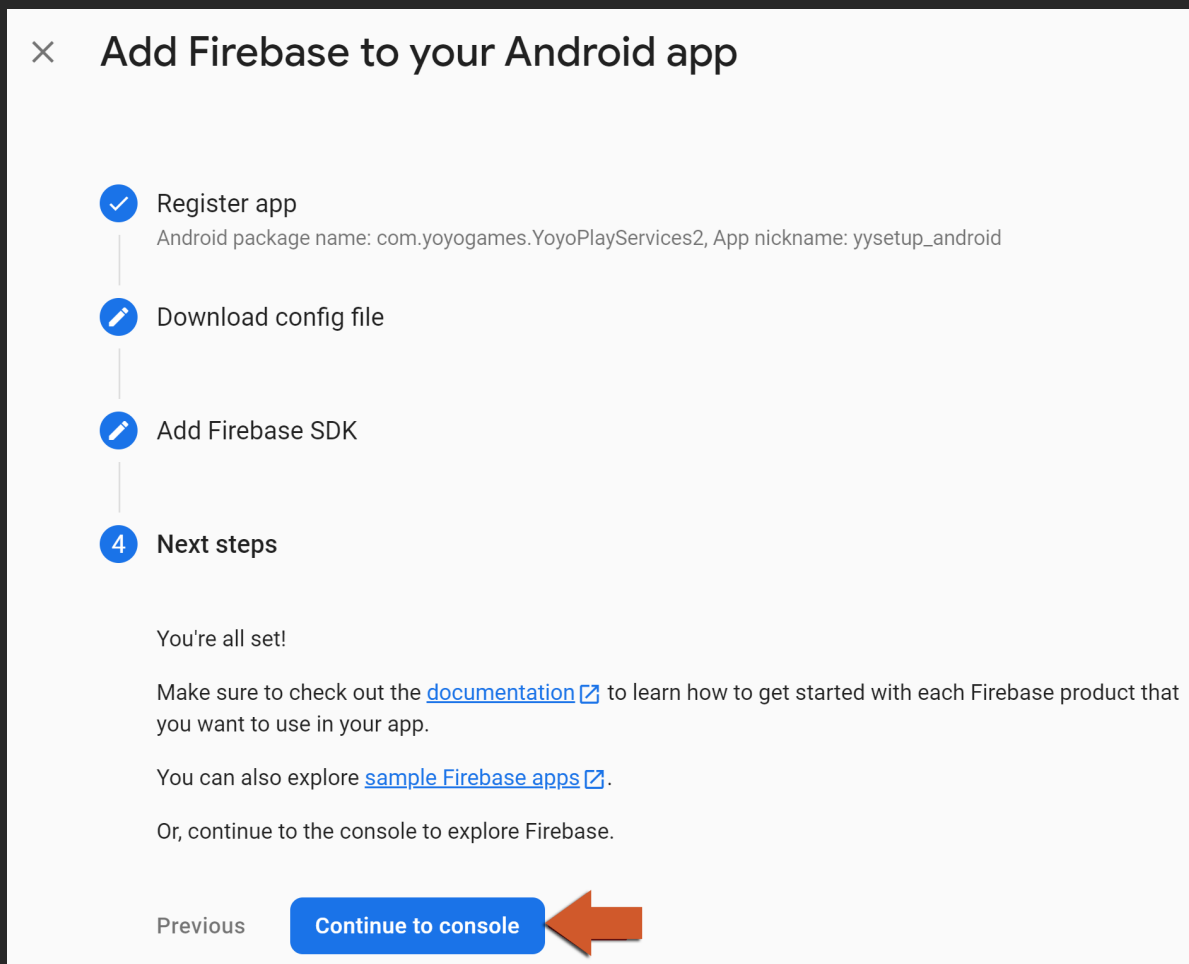
The Google services plugin for [Gradle](#) loads the google-services.json file you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (<project>/build.gradle):

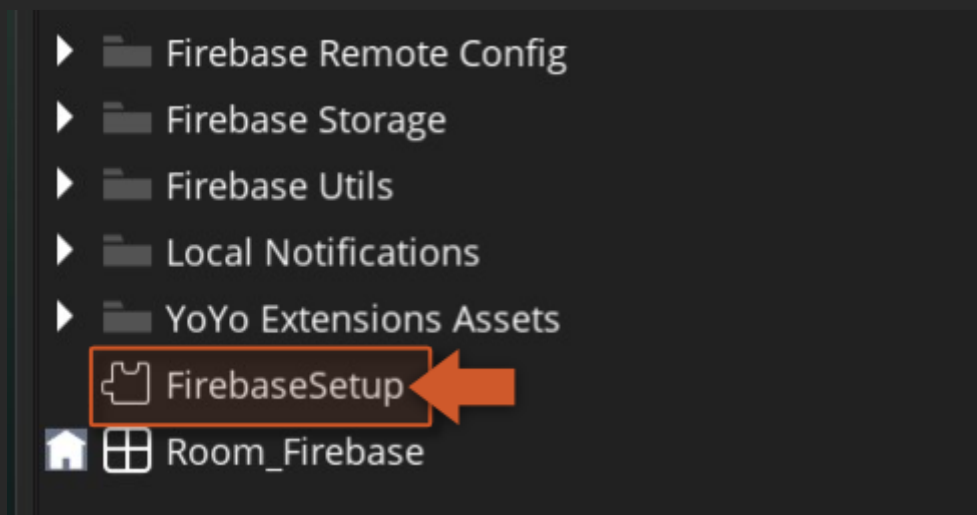
```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
    }
}
```



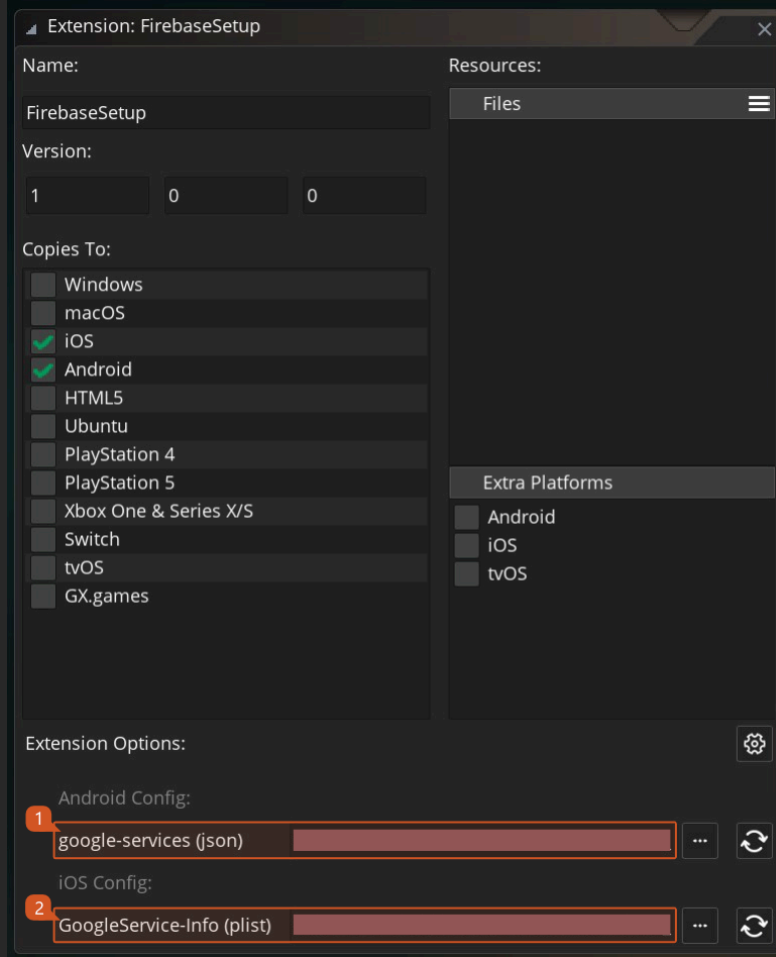
6. Click on the Continue to console button.



7. Now go into GameMaker, double click the extension **FirestoreSetup** asset.



8. In the extension panel just fill in the paths for the correct files (Android and/or iOS).



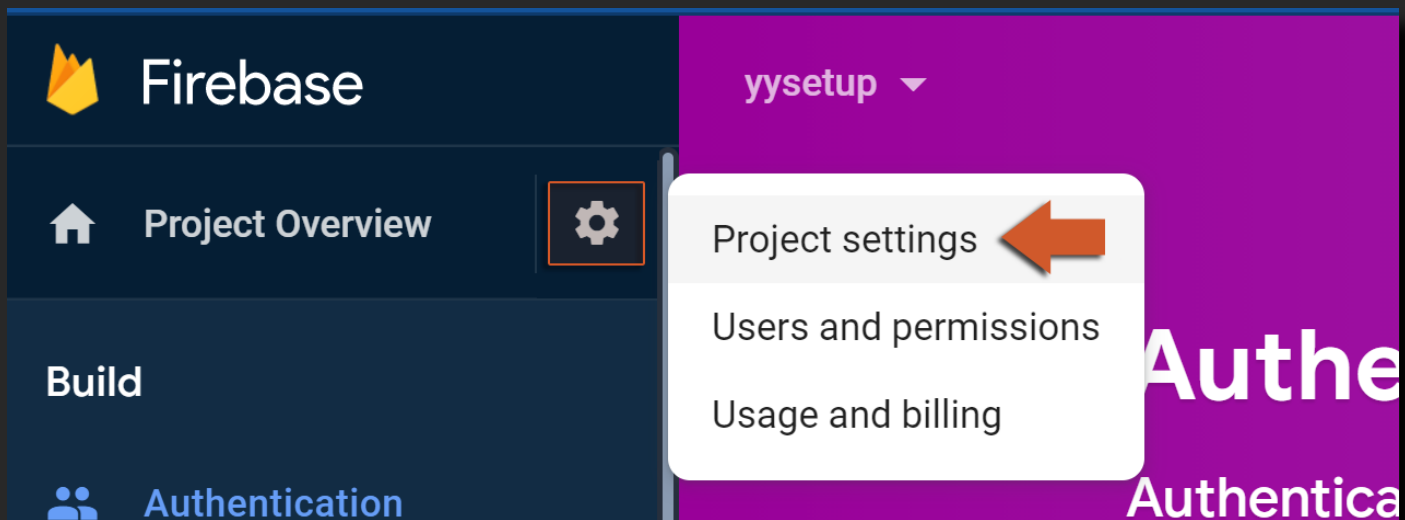
9. You have now finished the main setup for all Firebase Android modules!

iOS Setup

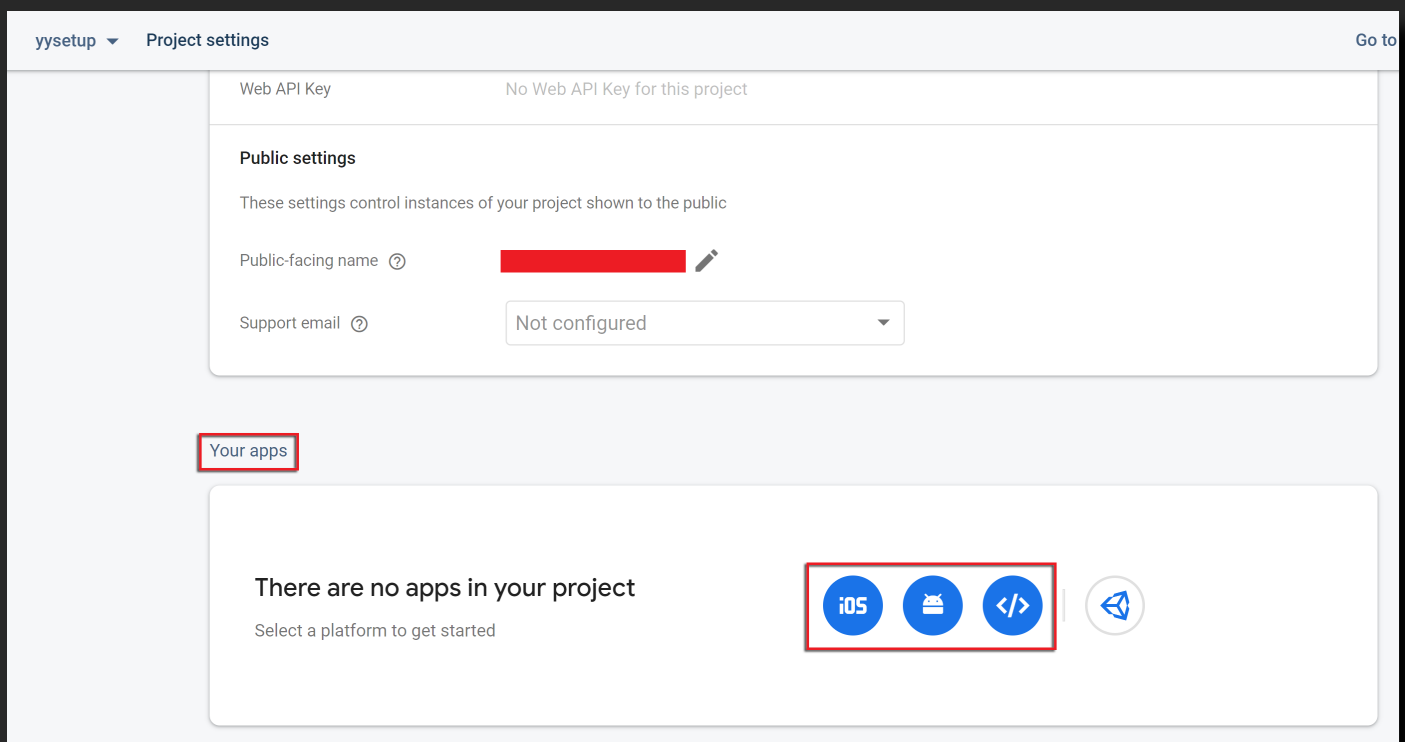
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the `GoogleServices-Info.plist` file into your project.

IMPORTANT Please refer to [this Helpdesk article](#) for instructions on setting up an iOS project.

1. Click the **Settings** icon (next to Project Overview) and select **Project settings**:



2. Now go to the **Your apps** section and click on the **iOS** button:




3. Fill the form with your iOS Bundle ID, App nickname and AppStore ID (last two are optional).

×

Add Firebase to your iOS app

1 Register app

iOS bundle ID ⓘ

com.company.appname

App nickname (optional) ⓘ

My iOS App

App Store ID (optional) ⓘ

123456789

Register app

4. Click on **Download GoogleService-info.plist** (make sure to save this file as we will need it in subsequent steps).

× Add Firebase to your iOS app

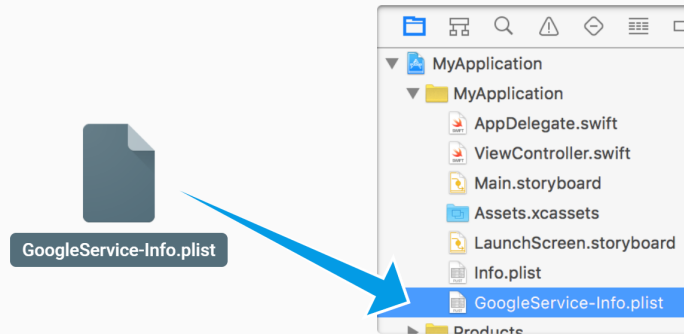
✓ Register app
iOS bundle ID: com.yoyogames.yyfirebase

2 Download config file

Instructions for Xcode below | [Unity](#) [C++](#)



Move the GoogleService-Info.plist file you just downloaded into the root of your Xcode project and add it to all targets.



Next

5. Ignore this screen, as this is already done in the extension.

× Add Firebase to your iOS app

✓ Register app
iOS bundle ID: com.yoyogames.yyfirebase

Download config file

3 Add Firebase SDK

Instructions for CocoaPods | [SwiftPM](#) [Download ZIP](#) [Unity](#) [C++](#)

Google services use [CocoaPods](#) to install and manage dependencies. Open a terminal window and navigate to the location of the Xcode project for your app.

Create a Podfile if you don't have one:

```
$ pod init
```



Open your Podfile and add:

```
# add the Firebase pod for Google Analytics
```

6. Ignore this screen as well, as this is also done in the extension.

× Add Firebase to your iOS app



Register app

iOS bundle ID: com.yoyogames.yyfirebase



Download config file



Add Firebase SDK



Add initialization code

To connect Firebase when your app starts up, add the initialization code below to your main `AppDelegate` class.

☒ Swift ☐ Objective-C

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
```



7. Click on the Continue to console button:

× Add Firebase to your iOS app



Register app

iOS bundle ID: com.yoyogames.yyfirebase



Download config file



Add Firebase SDK



Add initialization code



Next steps

You're all set!

Make sure to check out the [documentation](#) to learn how to get started with each Firebase product that you want to use in your app.

You can also explore [sample Firebase apps](#).

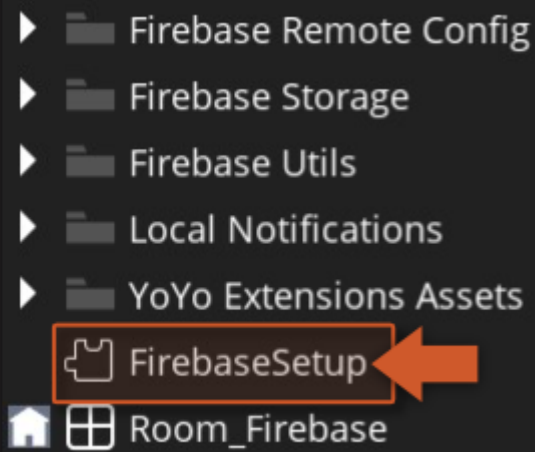
Or, continue to the console to explore Firebase.

Previous

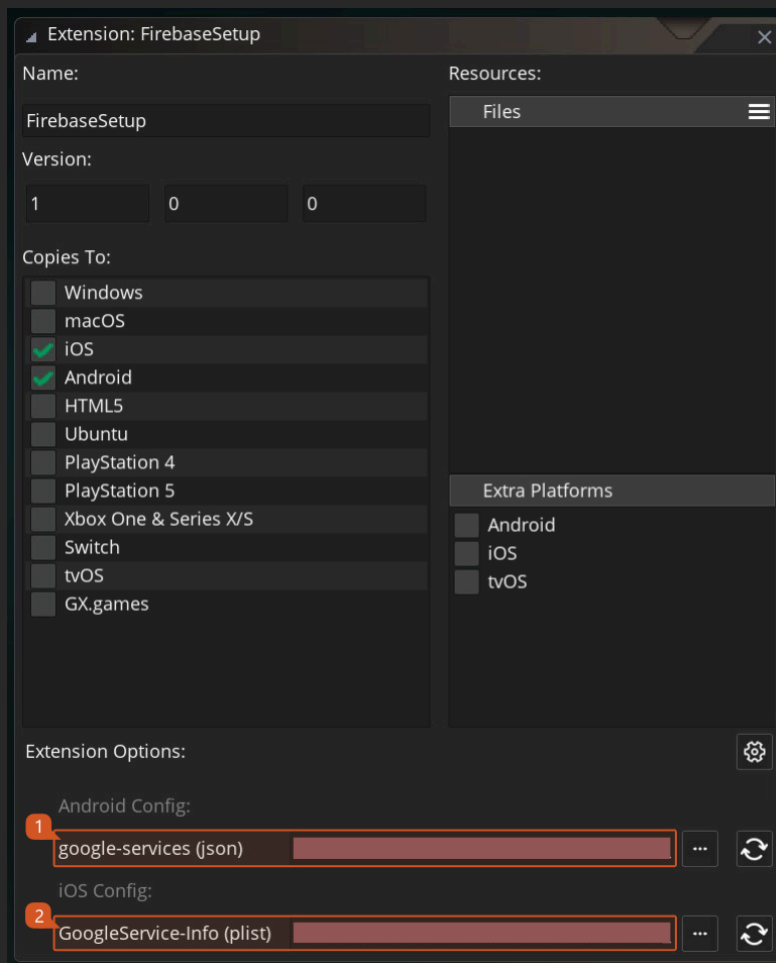
Continue to console



8. Now go into GameMaker, double click the extension **FirestoreSetup** asset.



9. In the extension panel just fill in the paths for the correct files (Android and/or iOS).



10. Make sure to set up **CocoaPods** for your project *unless* you are only using the REST API in an extension (if one is provided -- not all extensions provide a REST API) or the Firebase Cloud Functions extension (which only uses a REST API).

11. You have now finished the main setup for all Firebase iOS modules!

FirebaseCloudMessaging_DeleteToken

Deletes the FCM registration token for this Firebase project. Note that if auto-init is enabled, a new token will be generated the next time the app is started. Disable auto-init (using the function [FirebaseCloudMessaging_SetAutoInitEnabled](#)) to avoid this behavior.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirebaseCloudMessaging_DeleteToken();
```

Returns:

N/A

Triggers:

Asynchronous Social Event

Key	Type	Description
type	string	The constant <code>"FirebaseCloudMessaging_DeleteToken"</code>
success	boolean	Whether or not the function task succeeded.

Example:

```
FirebaseCloudMessaging_DeleteToken()
```

In the code above we request for the FCM token to be deleted. The function callback can be caught inside an **Async Social** event.

```
if(async_load["type"] == "FirebaseCloudMessaging_DeleteToken")
{
    if(async_load["success"])
    {
```

```
        show_debug_message("FCM token deleted")
    }
}
```

The code above matches the response against the correct event **type**, and provides a success message if **success** is true.

FirestoreCloudMessaging_GetToken

Requests the FCM registration token for this Firebase project. This sends information about the application and the device where it's running to the Firebase backend. See [FirestoreCloudMessaging_DeleteToken](#) for information on deleting the token.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirestoreCloudMessaging_GetToken();
```

Returns:

N/A

Triggers:

Asynchronous Social Event

Key	Type	Description
type	string	The constant <code>"FirestoreCloudMessaging_GetToken"</code>
success	boolean	Whether or not the function task succeeded.
value	string	The FCM registration token.

Example:

```
FirestoreCloudMessaging_GetToken()
```

In the code above we request for the current FCM token. The function callback can be caught inside an **Async Social** event.

```
if(async_load[?"type"] == "FirebaseCloudMessaging_GetToken")
{
    if(async_load[?"success"])
    {
        global.fcmToken = async_load[? "value"];
    }
}
```

The code above matches the response against the correct event **type**, and if the task succeeds it stores the token value into a global variable (`global.fcmToken`).

FirebaseCloudMessaging_IsAutoInitEnabled

Returns whether FCM auto-initialization is enabled or disabled.

Syntax:

```
Fi rebaseCl oudMessagi ng_I sAutoI ni tEnabl ed()
```

Returns:

Bool

Example:

```
i f (Fi rebaseCl oudMessagi ng_I sAutoI ni tEnabl ed())  
{  
    FirebaseCloudMessaging_SetAutoInitEnabled(false)  
}
```

The code above checks if auto-initialization is enabled and if it is it disables it (using [FirebaseCloudMessaging_SetAutoInitEnabled](#)).

FirebaseCloudMessaging_SetAutoInitEnabled

Enables or disables auto-initialization of Firebase Cloud Messaging.

When enabled, Firebase Cloud Messaging generates a registration token on app startup if there is no valid one (see [FirebaseCloudMessaging_GetToken](#)) and periodically sends data to the Firebase backend to validate the token. This setting is persistent across app restarts.

NOTE By default, Firebase Cloud Messaging auto-initialization is enabled.

Syntax:

```
Fi rebaseCl oudMessagi ng_SetAutoI ni tEnabl ed(enabl ed)
```

Argument	Type	Description
enabled	boolean	Whether auto-initialization should be turned on or off.

Returns:

N/A

Example:

```
i f (Fi rebaseCl oudMessagi ng_IsAutoI ni tEnabl ed())  
{  
    FirebaseCloudMessaging_SetAutoInitEnabled(false)  
}
```

The code above checks if auto-initialization is enabled (using the [FirebaseCloudMessaging_IsAutoInitEnabled](#) function) and if it is disables it.

FirestoreCloudMessaging_SubscribeToTopic

Subscribes the user to the given topic in the background. The subscription operation is persistent and will keep retrying until it is successful. This uses the FCM registration token to identify the app instance, generating one if it does not exist (see [FirestoreCloudMessaging_GetToken](#)), which periodically sends data to the Firebase backend when auto-init is enabled. To delete the data, delete the token (see [FirestoreCloudMessaging_DeleteToken](#)).

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirestoreCloudMessaging_SubscribeToTopic(topic)
```

Argument	Type	Description
topic	string	The name of the topic to subscribe to.

Returns:

N/A

Triggers:

Asynchronous Social Event

Key	Type	Description
type	string	The constant <code>"FirestoreCloudMessaging_SubscribeToTopic"</code>
success	boolean	Whether or not the function task succeeded.
value	string	The name of topic subscription requested.

Example:

```
Fi rebaseCl oudMessagi ng_Subscri beToTopi c("my_awesome_topi c")
```

In the code above we request a subscription to a topic (`"my_awesome_topi c"`). The function callback can be caught inside an **Async Social** event.

```
i f(async_l oad[?"type"] == "Fi rebaseCl oudMessagi ng_Subscri beToTopi c")
{
    var _topic = async_load[? "topic"];

    i f(async_load[?"success"])
    {
        show_debug_message(Subscription to " + _topic + " SUCCEEDED")
    }
    e lse
    {
        show_debug_message(Subscription to " + _topic + " FAILED")
    }
}
```

The code above matches the response against the correct event **type**, and if the task succeeds it stores the topic value in a local variable (`_topic`) and logs the success of the operation.

FirestoreCloudMessaging_UnsubscribeFromTopic

Unsubscribes from a previously subscribed topic (see [FirestoreCloudMessaging_SubscribeToTopic](#)) in the background. The unsubscribe operation is persistent and will keep retrying until it is completed.

This is an asynchronous function that will trigger the **Async Social** event when the task is finished.

Syntax:

```
FirestoreCloudMessaging_UnsubscribeFromTopic(topic)
```

Argument	Type	Description
topic	string	The name of the topic to unsubscribe from.

Returns:

N/A

Triggers:

Asynchronous Social Event

Key	Type	Description
type	string	The constant <code>"FirestoreCloudMessaging_UnsubscribeFromTopic"</code>
success	boolean	Whether or not the function task succeeded.
value	string	The name of topic subscription requested.

Example:

```
FirestoreCloudMessaging_UnsubscribeFromTopic("my_awesome_topic")
```

In the code above we request for the subscription to the topic `"my_awesome topic"` to be canceled. The function callback can be caught inside an **Async Social** event.

```
if(async_load[?"type"] == "FirebaseMessaging_UnsubscribeFromTopic")
{
    var _topic = async_load[? "topic"];

    if(async_load[?"success"])
    {
        show_debug_message(Subscription removed successfully)
    }
    else
    {
        show_debug_message(Subscription could not be removed)
    }
}
```

The code above matches the response against the correct event **type**, and if the task succeeds it stores the topic value in a local variable (`_topic`) and logs the success of the operation.