



# Firebase Crashlytics

Firebase Crashlytics is a lightweight, real-time crash reporter that helps you track, prioritize, and fix stability issues that erode your app quality. Crashlytics saves you troubleshooting time by intelligently grouping crashes and highlighting the circumstances that lead up to them.

Find out if a particular crash is impacting a lot of users. Get alerts when an issue suddenly increases in severity. Figure out which lines of code are causing crashes. All this information gets delivered to you through the online Firebase Console. Crashlytics also easily integrates into your Android, iOS, macOS, tvOS, and watchOS apps.

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

## Setup

Before starting to use any Firebase extensions, you are required to follow some general configuration steps; however the Crashlytics extension has some specific requirements which you can read in the pages below:

- [Create Project](#)
- [Platform Setup](#)
- [Building on iOS](#)

## Functions

The following functions are given for working with Firebase Crashlytics extension:

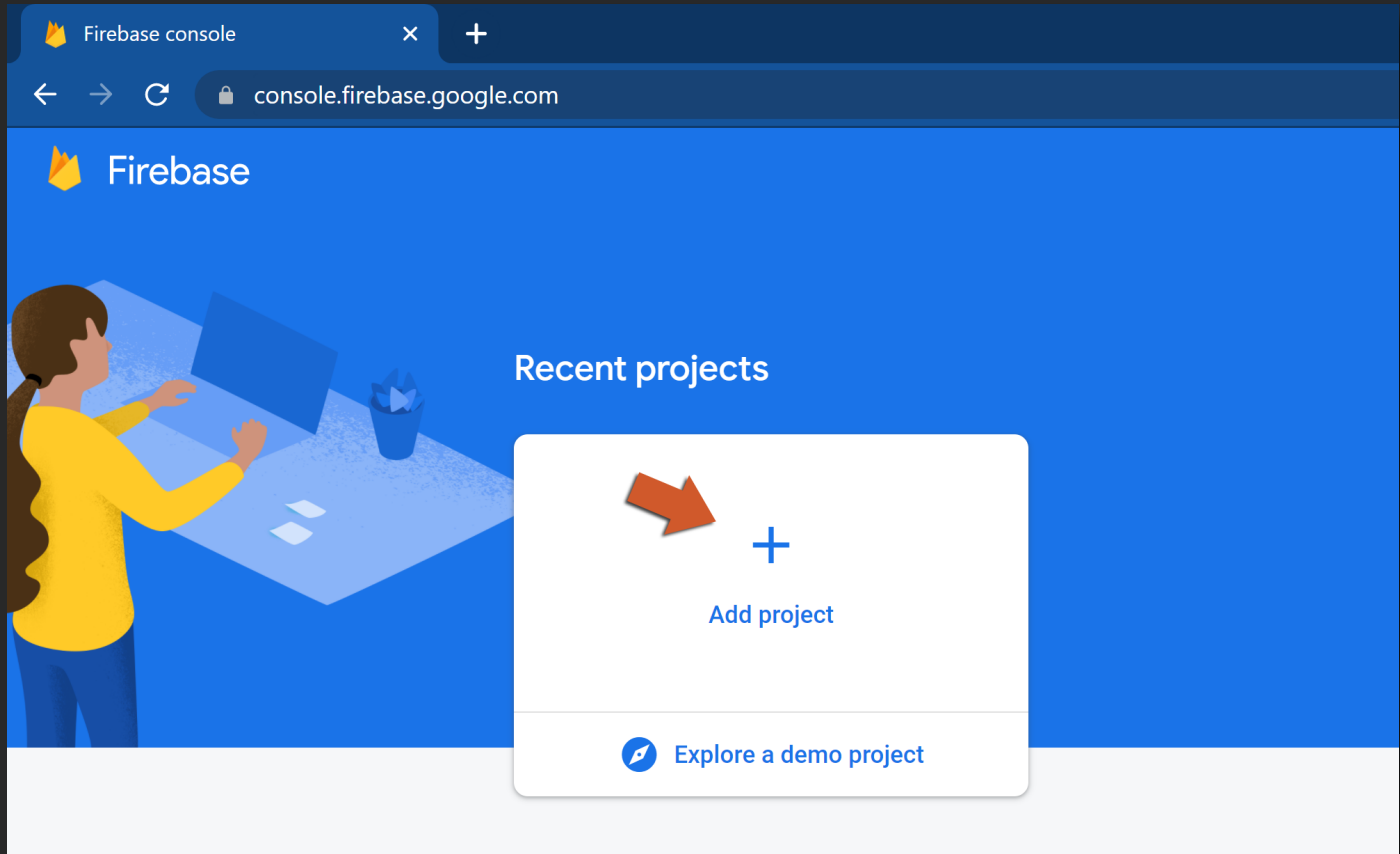
- [FirebaseCrashlytics\\_Crash](#)



# 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 <sup>?</sup>

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 <sup>?</sup>

 Crash-free users <sup>?</sup>

 User segmentation & targeting across  
Firebase products <sup>?</sup>

 Event-based Cloud Functions triggers <sup>?</sup>

 Predicting user behavior <sup>?</sup>

 Free unlimited reporting <sup>?</sup>



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 ?

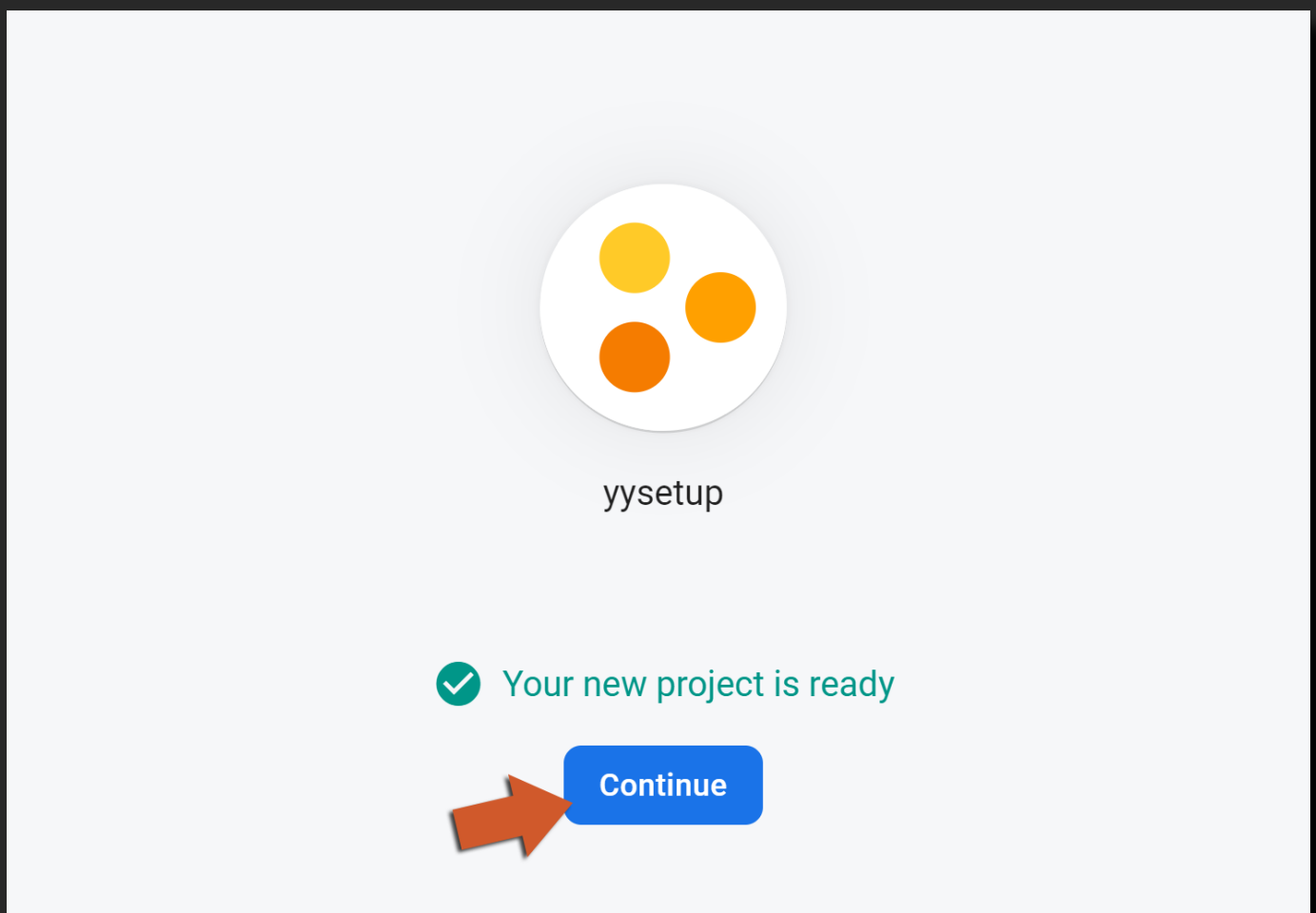
  Default Account for Firebase ▼

Automatically create a new property in this account 

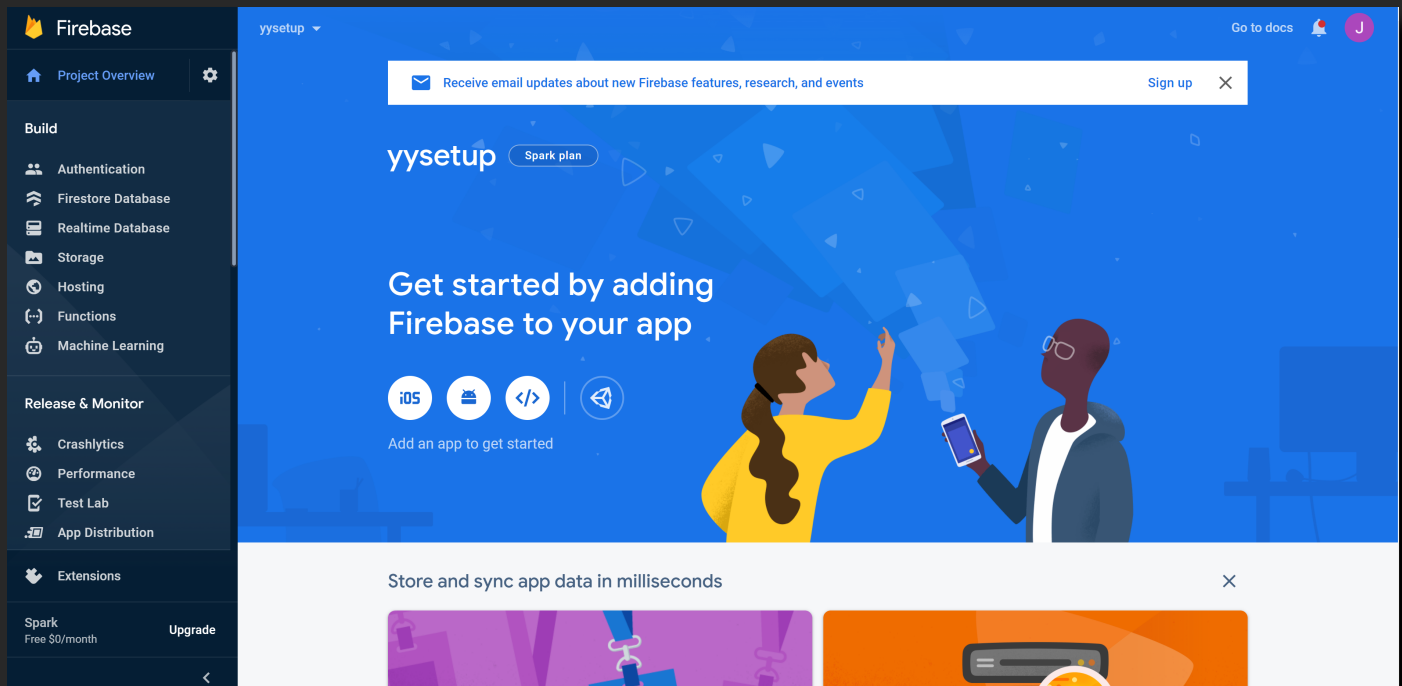
Upon project creation, a new Google Analytics property will be created in your chosen Google Analytics account and linked to your Firebase project. This link will enable data flow between the products. Data exported from your Google Analytics property into Firebase is subject to the Firebase terms of 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!

# Crashlytics Setup

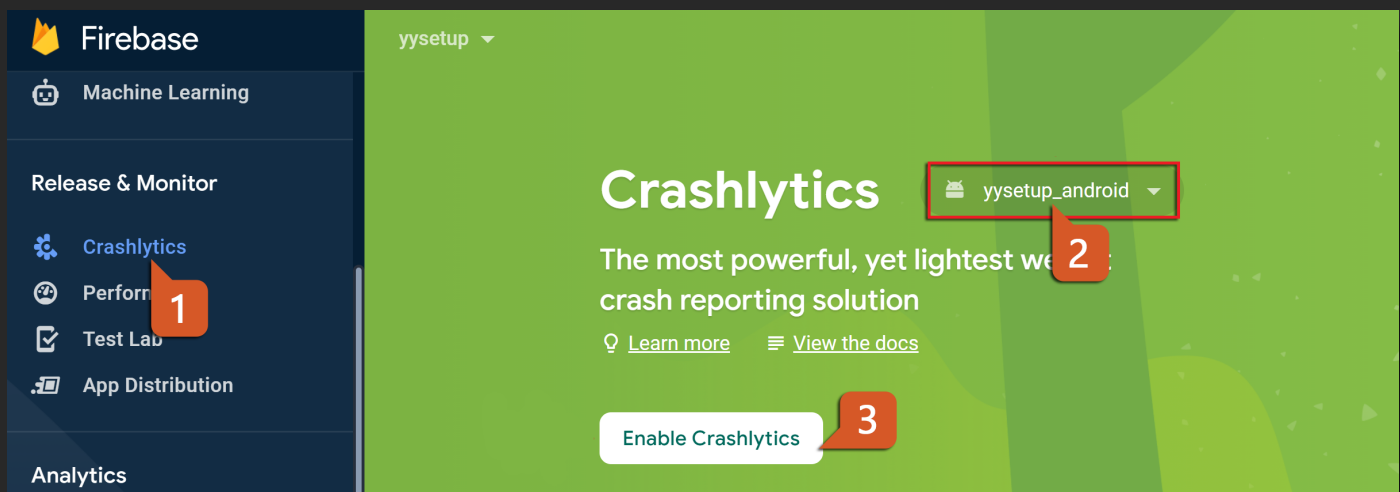
Firebase Crashlytics implementation uses SDK dependencies and therefore is only available on **Android** and **iOS** targets. In this section we will cover the steps necessary to start using the Firebase Crashlytics extension in your game.

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**

## Enabling Crashlytics

1. On your Firebase console, select **Crashlytics** under "Release & Monitor"; then select your **Android/iOS** project and click on **Enable Crashlytics**.



2. On this screen, the Firebase console will wait for a test crash from your app. To skip this screen you need to build a project (see **Building on iOS** if you are building for the iOS target), unplug your device and run a call to the included **FirebaseCrashlytics\_Crash** function.



## The most powerful, yet lightest weight crash reporting solution

💡 [Learn more](#)

☰ [View the docs](#)



Add the Firebase Android SDK or Unity Plugin.

We'll be listening for your app's first crash!

[View the docs](#) ↗

Need help with your first crash? [Learn how to implement a test crash](#)

3. You are now all set to monitor your game crash logs.

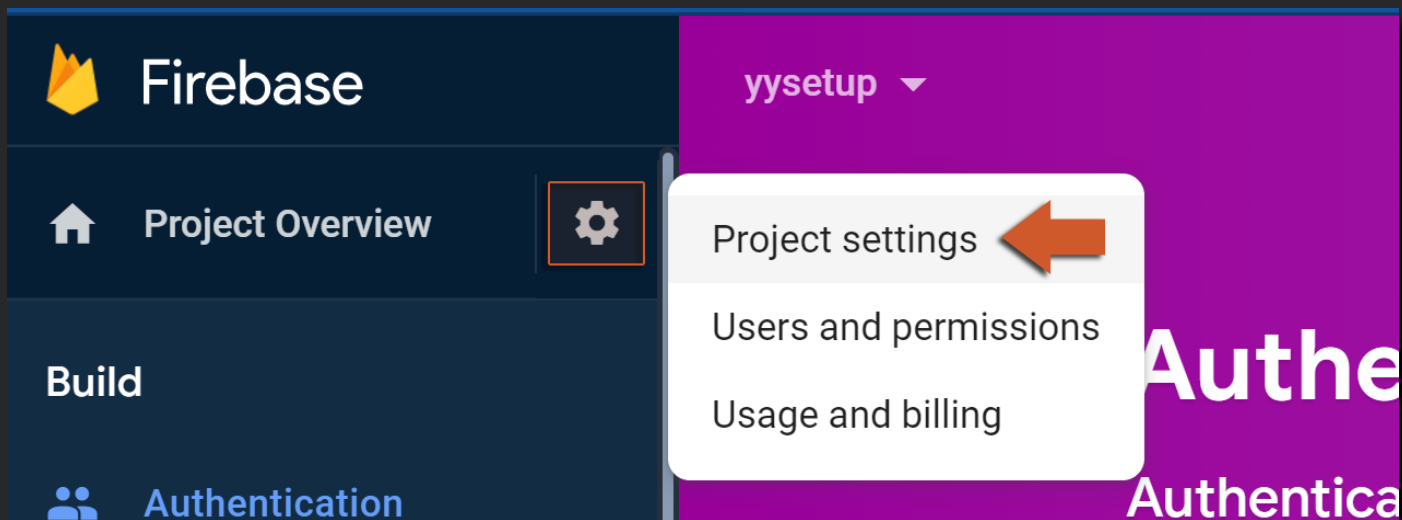


# Android Setup

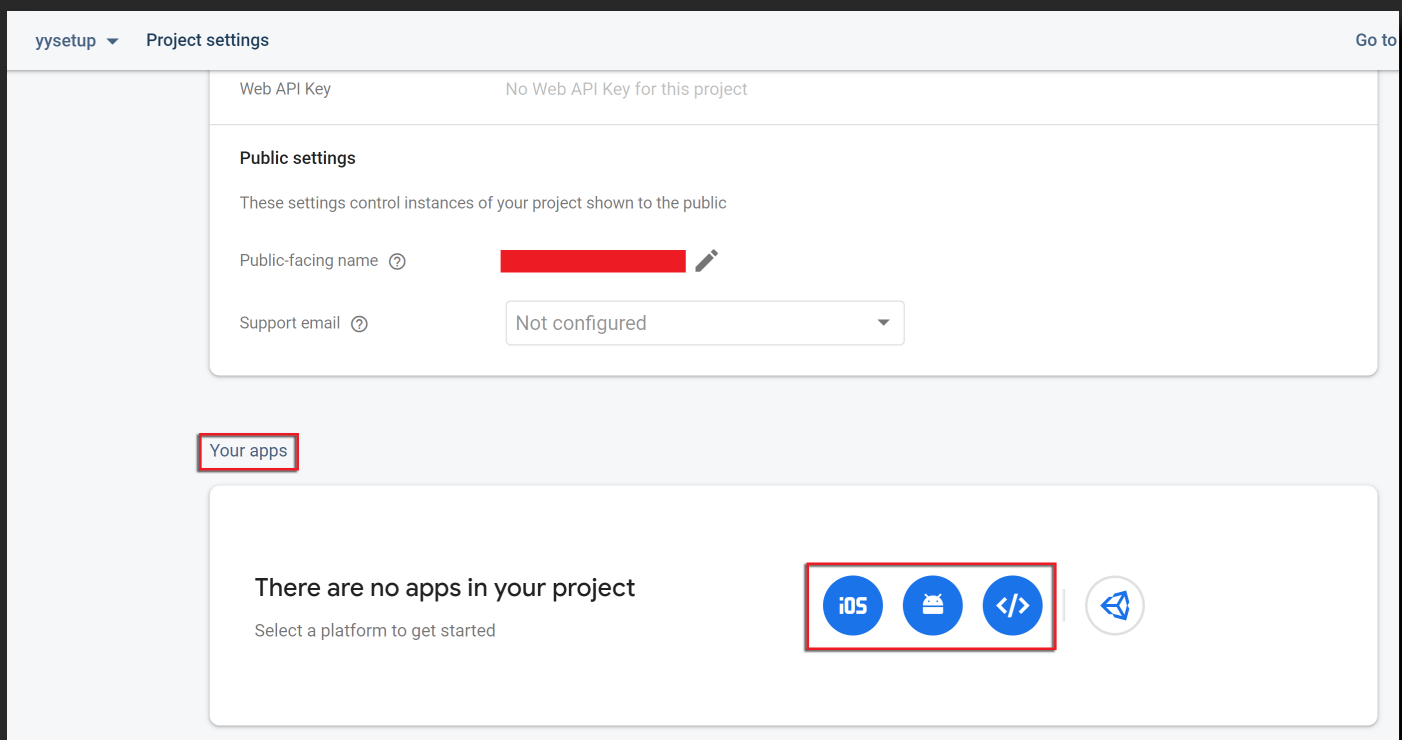
This setup is necessary for all the Firebase modules using Android and needs to be done once per project, and basically involves importing the `google-services.json` file into your project.

**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

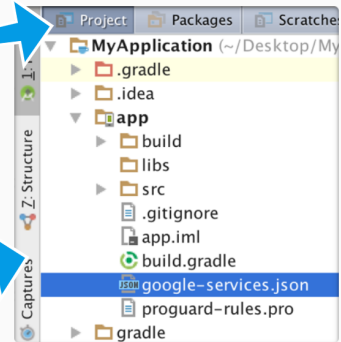
Switch to the Project view in Android Studio to see your project root directory.

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++](#)

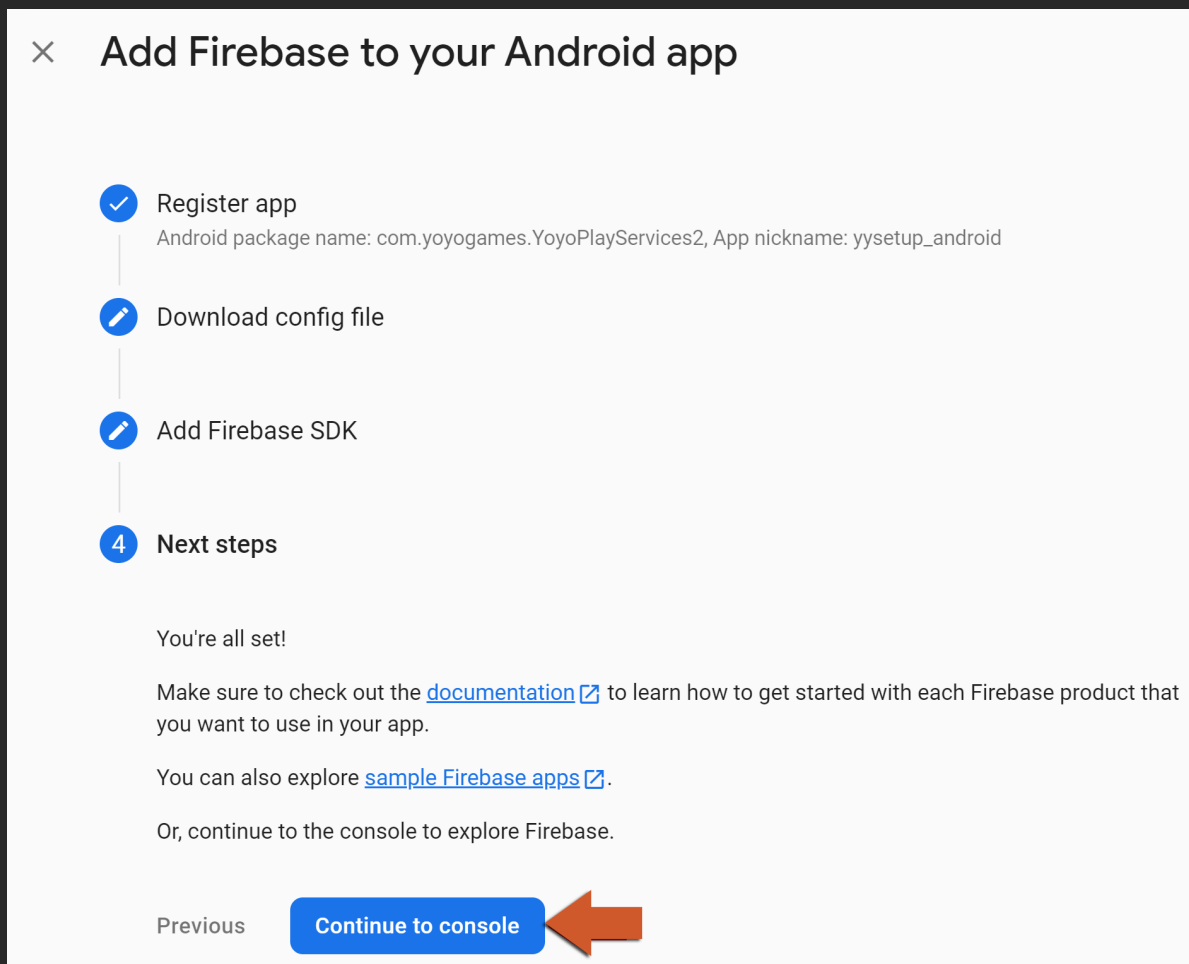
The Google services plugin for [Gradle](#) loads the google-services.json file you just downloaded. Modify your build.gradle files to use the plugin.

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

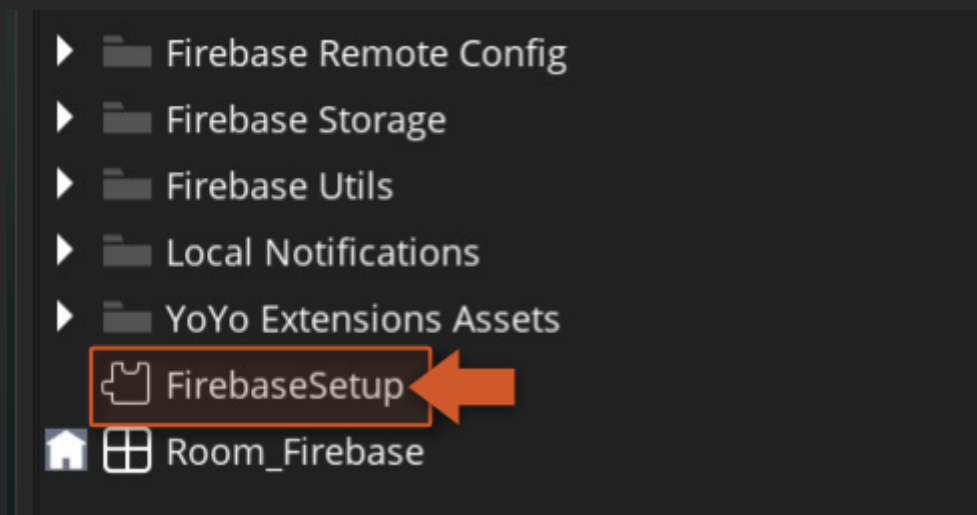
```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
    }
}
```



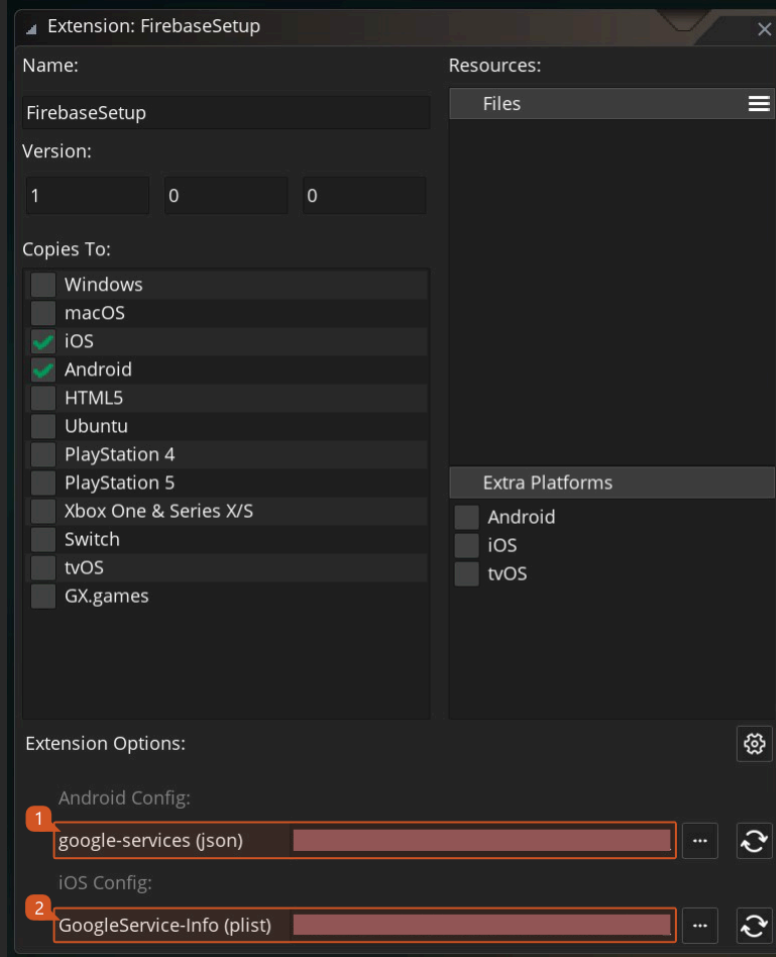
6. Click on the Continue to console button.



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).



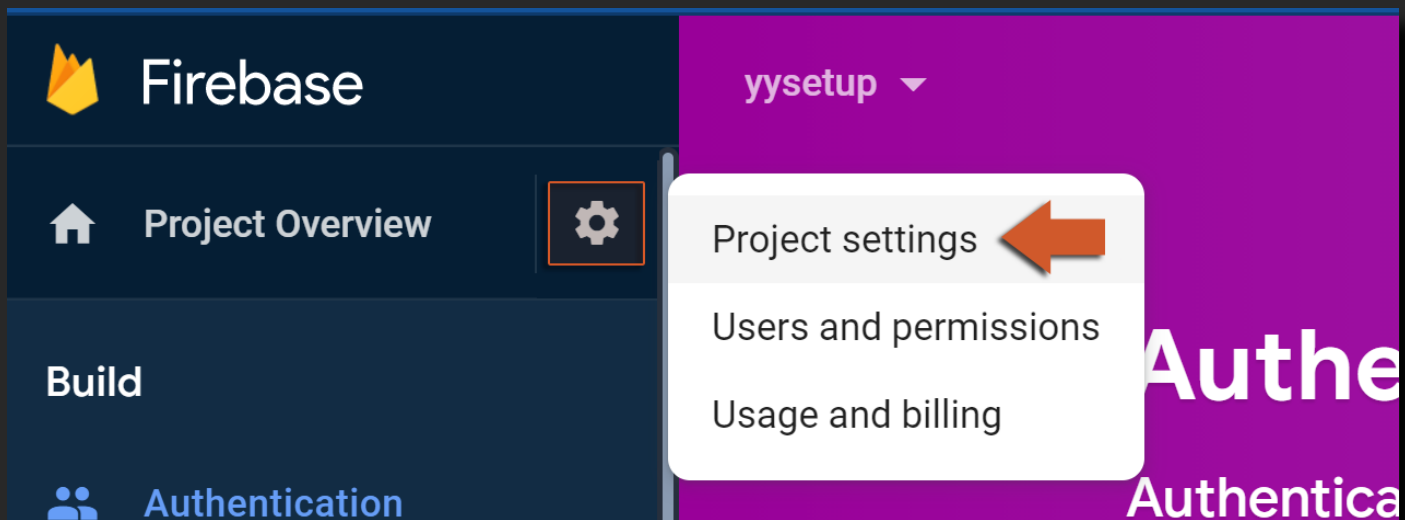
9. You have now finished the main setup for all Firebase Android modules!

# iOS Setup

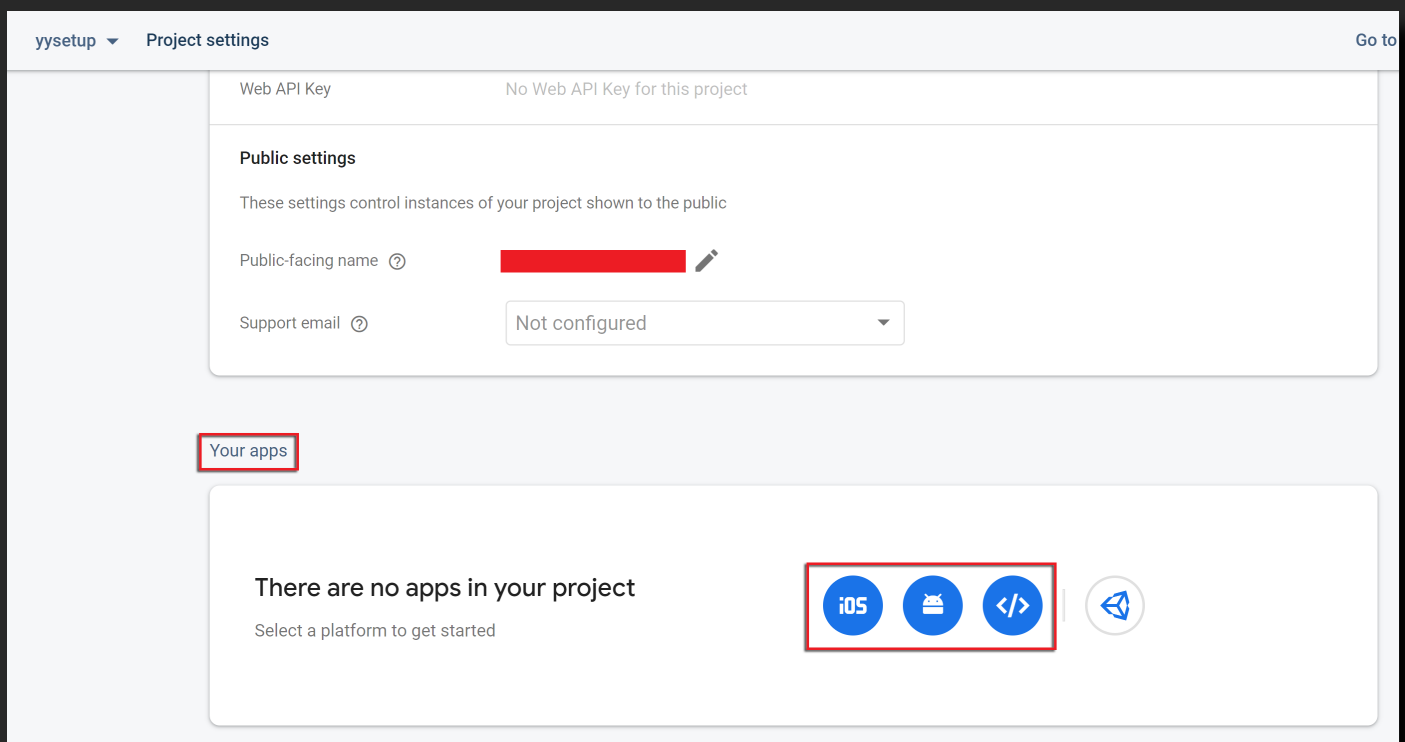
This setup is necessary for all the Firebase modules using iOS and needs to be done once per project, and basically involves importing the `GoogleServices-Info.plist` file into your project.

**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

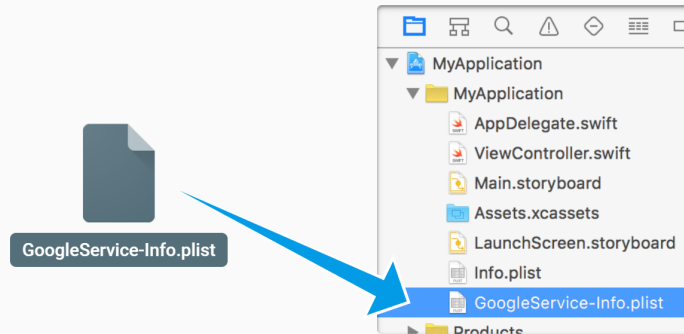
✓ Register app  
iOS bundle ID: com.yoyogames.yyfirebase

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

4 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

5 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](#).

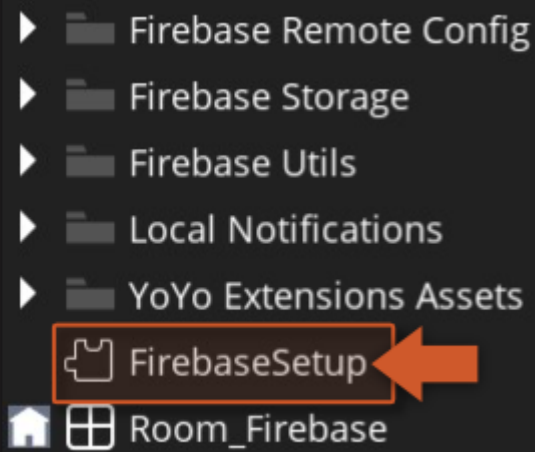
Or, continue to the console to explore Firebase.

Previous

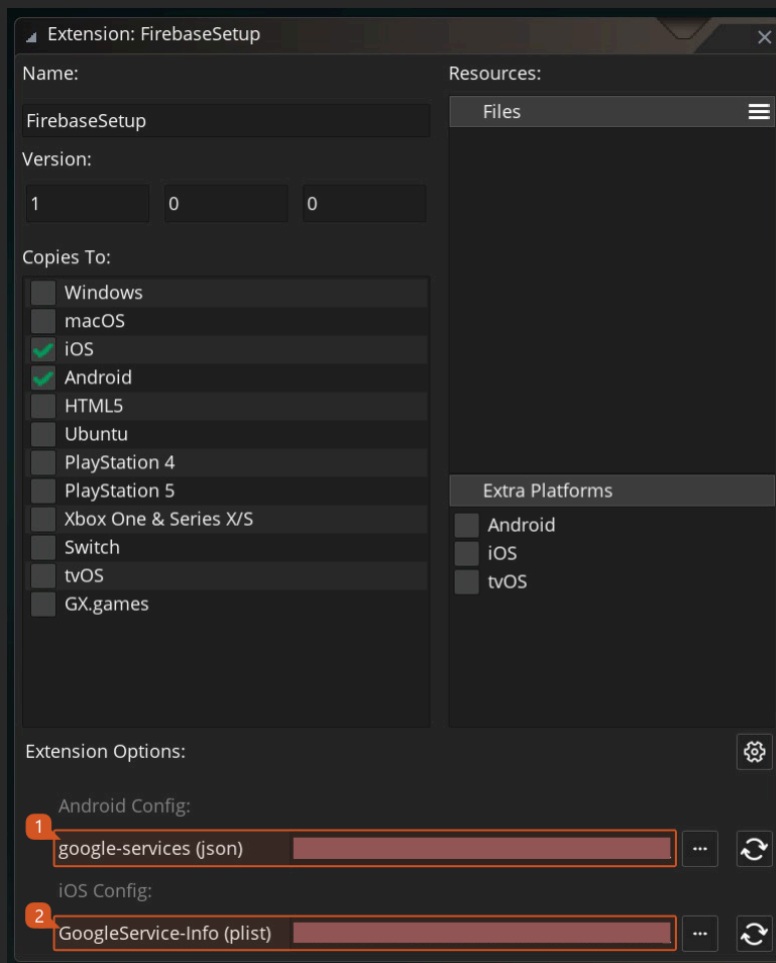
Continue to console



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).



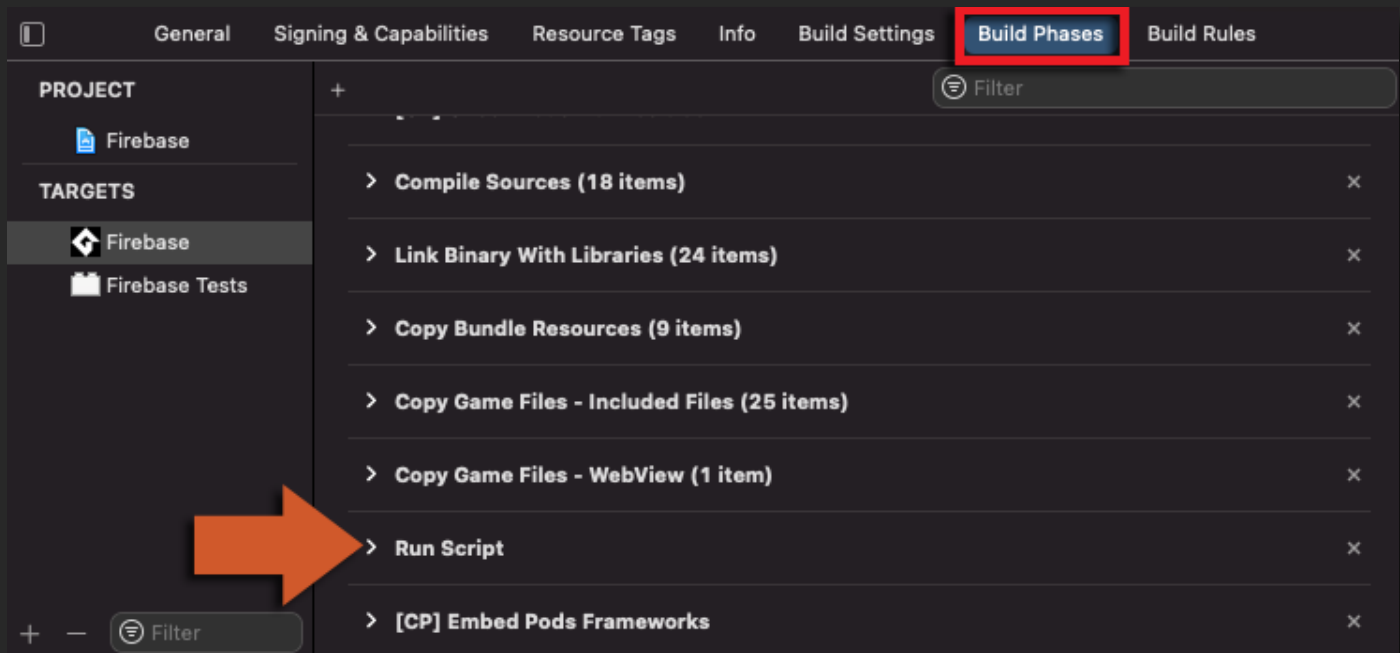
10. Make sure to set up **CocoaPods** for your project *unless* you are only using the REST API in an extension (if one is provided -- not all extensions provide a REST API) or the Firebase Cloud Functions extension (which only uses a REST API).

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

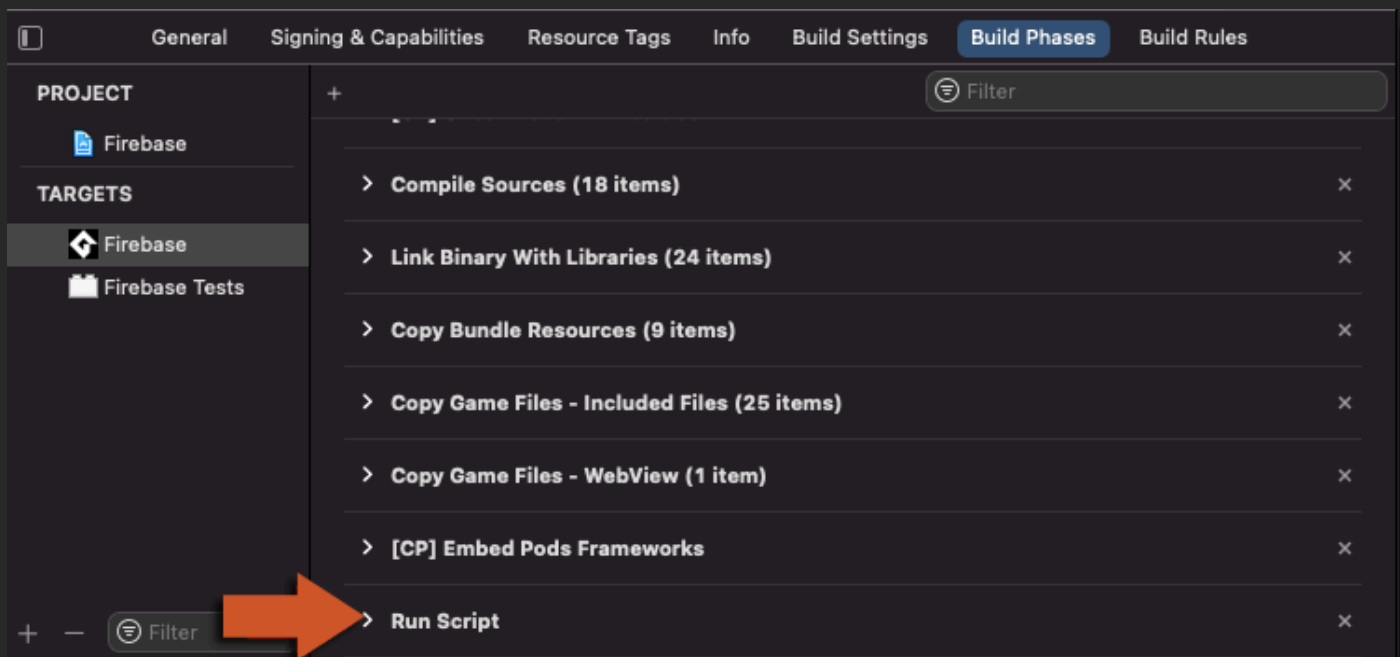
# Building on iOS

Building your project on iOS requires some Xcode project changes:

1. Build your project from GMS2 first (using the iOS target) so it creates an Xcode project that we will edit in the steps below.
2. Before building your project further from Xcode, go to **Build Phases** and drag the **Run Script** item to the bottom:



3. The result should look like this (with **Run Script** at the bottom of the list):



4. You can now continue with the building process and the application will work as expected.



# FlutterCrashlytics\_Crash

This function throws an error that crashes the app and creates a fatal report to be sent to Crashlytics server. A crash created using this function can be tracked during the next application execution using the method `FlutterCrashlytics_DidCrashOnPreviousExecution`.

**IMPORTANT** This function needs to be called at least once to enable Crashlytics for the first time.

## Syntax:

```
FlutterCrashlytics_Crash(errorMessage);
```

Argument	Type	Description
errorMessage	string	The error message to report (this will be a <b>fatal</b> report).

## Returns:

N/A

## Example:

```
FlutterCrashlytics_Crash("Test Crash");
```

The code above logs a **fatal error** report to the Crashlytics server. Another way to deal with error reporting would be to call the `exception_unhandled_handler` function and override the default error handling function with the example below.

```
exception_unhandled_handler(function(_exception) {  
    var _info = json_stringify(_exception);  
  
    FlutterCrashlytics_Crash(_info);  
    show_debug_message(_info);  
});
```

The code above catches all the exceptions and trigger a **fatal error** report with the information from the exception converted to a string (using the function `json_stringify`) and it will also print the same information to the debug console.

## FirebaseCrashlytics\_CrashlyticsCollectionEnabled\_Check

This function returns whether automatic data collection is enabled or not. The function uses two ways to check whether automatic data collection is enabled, in order of priority:

- If **FirebaseCrashlytics\_CrashlyticsCollectionEnabled\_Set** was called, uses its value.
- If the `Fi rebaseCrashl yti csCol l ecti onEnabl ed` key is in your app's `Info. pl i st`, uses that.

**IMPORTANT** This function is only available on **iOS** targets.

### Syntax:

```
Fi rebaseCrashl yti cs_Crashl yti csCol l ecti onEnabl ed_Check()
```

### Returns:

Bool

### Example:

```
if (Fi rebaseCrashl yti cs_Crashl yti csCol l ecti onEnabl ed_Check())  
{  
    FirebaseCrashlytics_CrashlyticsCollectionEnabled_Set(false);  
}
```

The code above checks if Crashlytics automatic data collection is enabled and if so it will disable it (using the function **FirebaseCrashlytics\_CrashlyticsCollectionEnabled\_Set**).

# Flutter FirebaseCrashlytics\_CrashlyticsCollectionEnabled\_Set

This function is used to enable or disable the automatic data collection configuration for Crashlytics. If this is set, it overrides any automatic data collection settings configured in the extension options as well as any Firebase-wide settings.

If automatic data collection is disabled for Crashlytics, crash reports are stored on the device. To check for unsent reports, use the [Flutter FirebaseCrashlytics\\_UnsentReports\\_Check](#) function. Use [Flutter FirebaseCrashlytics\\_UnsentReports\\_Send](#) to upload existing reports when automatic data collection is disabled. Use [Flutter FirebaseCrashlytics\\_UnsentReports\\_Delete](#) to delete any reports stored on the device without sending them to Crashlytics.

## Syntax:

```
Flutter FirebaseCrashlytics_CrashlyticsCollectionEnabled_Set(enabled)
```

Argument	Type	Description
enabled	boolean	Whether to enable automatic data collection.

## Returns:

N/A

## Example:

```
if (Flutter FirebaseCrashlytics_CrashlyticsCollectionEnabled_Check())  
{  
    Flutter FirebaseCrashlytics_CrashlyticsCollectionEnabled_Set(false);  
}
```

The code above checks if automatic data collection is enabled (using [Flutter FirebaseCrashlytics\\_CrashlyticsCollectionEnabled\\_Check](#)) and if so it will disable it.



# FirestoreCrashlytics\_DidCrashOnPreviousExecution

This function returns whether the app crashed during its previous run (any errors handled by the [FirestoreCrashlytics\\_Crash](#) function are considered to be crashes by this function).

## Syntax:

```
FirestoreCrashlytics_DidCrashOnPreviousExecution()
```

## Returns:

```
boolean
```

## Example:

```
if (FirestoreCrashlytics_DidCrashOnPreviousExecution())  
{  
    FirestoreCrashlytics_UnsentReports_Send();  
}
```

The code above checks if the application crashed on its previous execution and in that case, sends all the unsent reports to the Crashlytics servers (using the function [FirestoreCrashlytics\\_UnsentReports\\_Send](#)).

# FirestoreCrashlytics\_Log

This function logs a message that is included in the next fatal or non-fatal report. Logs are visible in the session view on the Firebase Crashlytics console. Newline characters are stripped and extremely long messages are truncated. The maximum log size is 64k. If exceeded, the log rolls such that messages are removed, starting from the oldest.

## Syntax:

```
FirestoreCrashlytics_Log(message)
```

Argument	Type	Description
message	string	The message to be logged.

## Returns:

N/A

## Example:

```
FirestoreCrashlytics_Log("Entered the " + room_get_name(room) + " room");
```

The code example above will log a message that will be included in the next report; this might be useful for identifying the source of the error within the game (e.g. in the example above we log the name of the current room so we know which room the error originated in).

# FirebaseCrashlytics\_RecordException

Records a non-fatal report to send to Crashlytics. Non-fatal reports are manually thrown by the developer and may not represent crashes that occurred during code execution.

## Syntax:

```
firebaseCrashlytics_RecordException(errorMessage)
```

Argument	Type	Description
errorMessage	string	Error message to include in the report.

## Returns:

N/A

## Example:

```
switch (characterClass)
{
    case "hero":
        // Do something
        break;

    case "enemy":
        // Do something else
        break;

    default:
        // There are no more classes, so we shouldn't be hitting this line
        firebaseCrashlytics_RecordException("characterClass has an unknown value
of : " + characterClass);
        break;
}
```

In the code above we have a switch statement where the variable ( `characterClass` ) is expected to only be equal to either `"hero"` or `"enemy"` . If it happens to be none of these then we might want to report this even though it's not a fatal error.

# FirebaseCrashlytics\_SetCustomKey

This function sets a custom key-value pair that is associated with subsequent fatal and non-fatal reports. Calling this function with an existing key will simply update the value for that key. The value of any key at the time of a fatal or non-fatal event is associated with that event. Keys and associated values are visible in the session view on the Firebase Crashlytics console.

Crashlytics supports a maximum of 64 key/value pairs. New keys beyond that limit are ignored. Keys or values that exceed 1024 characters are truncated.

## Syntax:

```
firebaseCrashlytics_SetCustomKey(key, value)
```

Argument	Type	Description
key	string	A unique key.
value	real/string	The value to be associated with the given key.

## Returns:

N/A

## Example:

```
firebaseCrashlytics_SetCustomKey("level", room_get_name(room));
```

The code above sets the custom key `"level"` to be included in subsequent reports, which can help the developer know which room caused a certain crash.

# FirebaseCrashlytics\_SetUserIdentifier

This function records a user ID (identifier) that is associated with subsequent fatal and non-fatal reports. The user ID is visible in the session view on the Firebase Crashlytics console. Identifiers longer than 1024 characters will be truncated.

## Syntax:

```
Fi rebaseCrashl yti cs_SetUserI denti fi er(i denti fi er)
```

Argument	Type	Description
identifier	string	A unique identifier for the current user.

## Returns:

N/A

## Example:

```
Fi rebaseCrashl yti cs_SetUserI denti fi er("MyUser1234567");
```

The code above will add additional user information to the any subsequent reports.

# FirebaseCrashlytics\_UnsentReports\_Check

This function checks the device for any fatal or non-fatal crash reports that haven't yet been sent to Crashlytics. If automatic data collection is enabled, then reports are uploaded automatically and this always returns false in its Async event. If automatic data collection is disabled, this function can be used to check whether the user opted in to send crash reports from their device.

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

**IMPORTANT** If there are any unsent reports, this function will only (asynchronously) return `true` for the first time it is called in each game execution.

## Syntax:

```
Fi rebaseCrashl yti cs_UnsentReports_Check()
```

## Returns:

N/A

## Triggers:

Asynchronous Social Event

Key	Type	Description
type	string	The string <code>"Fi rebaseCrashl yti cs_UnsentReports_Check"</code>
value	boolean	Whether or not there are unsent reports.

## Example:

```
Fi rebaseCrashl yti cs_UnsentReports_Check()
```

In the code above we are checking for collected unsent reports. The function callback can be caught inside an **Async Social** event.

```
if (async_load[? "type"] == "Fi rebaseCrashl yti cs_UnsentReports_Check")
{
    if (async_load[? "value"])
    {
        FirebaseCrashlytics_UnsentReports_Send();
    }
}
```

The code above matches the response against the correct event **type**, and if there are unsent reports we proceed to send them to the Crashlytics servers (using the function **FirebaseCrashlytics\_UnsentReports\_Send**).

# FirestoreCrashlytics\_UnsentReports\_Delete

If automatic data collection is disabled, this function queues up any unsent reports on the device for deletion. Otherwise, it does nothing.

## Syntax:

```
FirestoreCrashlytics_UnsentReports_Delete()
```

## Returns:

N/A

## Example:

```
if (async_load[? "type"] == "FirestoreCrashlytics_UnsentReports_Check")
{
    if (async_load[? "value"])
    {
        FirestoreCrashlytics_UnsentReports_Delete();
    }
}
```

The code above exists in the **Async Social** event as a callback for the **FirestoreCrashlytics\_UnsentReports\_Check** function, and checks if there are any unsent reports on the device; in that case, it deletes them.



# FirebaseCrashlytics\_UnsentReports\_Send

If automatic data collection is disabled, this function queues up any unsent reports on the device to be sent to Crashlytics. Otherwise, it does nothing.

## Syntax:

```
FirebaseCrashlytics_UnsentReports_Send()
```

## Returns:

N/A

## Example:

```
if (async_load[? "type"] == "FirebaseCrashlytics_UnsentReports_Check")
{
    if (async_load[? "value"])
    {
        FirebaseCrashlytics_UnsentReports_Send();
    }
}
```

The code above exists in the **Async Social** event as a callback for the **FirebaseCrashlytics\_UnsentReports\_Check** function, and checks if there are any unsent reports on the device; in that case, it proceeds to send them to the Crashlytics server.