UNREAL DEV GRANT RECIPIENT

GAMEDNA

ULTIMATE MOBILE KIT

FOR UNREAL ENGINE 4

Contents

1. License 6
2. Introduction 7
3. Plugin updates 8
a. Update from previous versions to 1.12.0 8
4. Example project 9
5. GitHub Repository 9
6. Getting started 10
7. Mobile Analytics 13
a. Initialization 15
b. Log Events 15
c. Set Current Screen 17
d. Set User Property 18
e. Set User ID 18
8. Cloud Messaging 19
a. Initialization 20
b. Access the device registration token 20
c. Subscribe/Unsubscribe 21
d. Receive messages 22
e. Send downstream messages 22
f. Send upstream messages 23
9. Push Notifications 24
a. Enable Push Notifications 25
b. Send a message from the Notifications console 25
10. Authentication 26
a. Initialization 27
b. Create User 28
i. Create User With Email And Password 28
ii. Send Email Verification 28
iii. Send Password Reset Email 29
c. Verify Phone Number 29
d. Sign In With Google 31

e. Link with external providers 32
i. Link With Credential 32
ii. Unlink Provider 33
iii. Fetch Providers for Email 33
f. Sign In 34
i. Sign In With Email And Password 34
ii. Sign In Anonymously 34
iii. Sign In With Credential 35
iv. Sign In With Custom Token 36
v. Get User Token 36
vi. Examples of Sign In/Link With Credential 37
g. Managing Users 38
i. Is User Logged In 38
ii. Get Logged User 38
iii. Sign Out 39
iv. Reauthenticate User 39
v. Reload User 40
vi. Delete User 40
h. Update User Profile 41
i. Update Email 41
ii. Update Password 41
iii. Update User Profile 42
i. Authentication Listener 42
j. User ID Token Changed Listener 43
11. Instance Id 44
a. Initialization 45
b. Instance Id 45
c. Tokens 46
12. Cloud Storage 47
a. Initialization 48
b. Storage Reference 48
c. Storage Path 50
d. Download File 50
e. Upload File 51

f. Delete File 52
g. Monitor progress of Downloads/Uploads 52
h. Manage Downloads/Uploads 53
i. Get/Update Metadata 53
13. Remote Config 55
a. Initialization 56
b. Get parameter values to use in your game 57
c. Set parameter values in the service (as needed) 58
d. Fetch and activate values from the service (as needed) 58
14. Performance Monitoring 60
15. Crashlytics 62
a. Force a crash to test your implementation 63
b. Customize crash reports 63
i. Add custom logs 63
ii. Add custom keys 64
iii. Set user data 64
c. Upload dSYMs symbols 64
i. Android 65
ii. iOS 65
16. Dynamic Links 66
a. Initialization 67
b. Create Dynamic Links 67
i. Create a long Dynamic Link 68
ii. Create a short Dynamic Link 68
c. Receive Dynamic Links 69
17. Invites 70
a. Initialization 72
b. Send Invitation 72
c. Receive Invitation 73
18. In-App Messaging 75
19. Test Lab 76
20. Cloud Functions 79
21. Predictions 82

22. A/B Testing 84
23. Hosting 85
24. C++ implementation 87

1. License

gameDNA Ltd grants you a non-exclusive, non-transferable, non-sublicensable license for a single User to use, reproduce, display, perform, and modify the Ultimate Mobile Kit for Unreal® Engine 4 for any lawful purpose (the "License"). The License becomes effective on the date you buy Ultimate Mobile Kit for Unreal® Engine 4. The License does not grant you any title or ownership in the Licensed Technology.

You may Distribute the Ultimate Mobile Kit for Unreal® Engine 4 incorporated in object code format only as an inseparable part of a Product to end users. The Product may not contain any Plugin Content in uncooked source format.

2. Introduction

Ultimate Mobile Kit is a plugin for Unreal Engine 4 that lets you integrate the **Firebase** platform for iOS & Android.

Firebase is a platform that helps you quickly develop high-quality experiences, grow your user base, and earn more money. **Firebase** is made up of complementary features that you can mix-and-match to fit your needs. You can focus on making your game and not waste time building complex infrastructure.

Current plugin version: 1.12.0

Support: support@gamednastudio.com

FEATURES:

- **Mobile Analytics** - heart of Firebase, see user behavior and measure attribution from a single dashboard.
- **Cloud Messaging** - lets you reliably deliver and receive messages at no cost.
- **Push Notifications** - schedule and send from dashboard unlimited notifications to engage the right players at the most relevant time.
- **Authentication** - complete authentication system that supports email & password, Facebook, Twitter, GitHub, Google, Google Play, Game Center, and Phone Number Authentication.
- **Instance Id** - generate unique user IDs for authentication and security tokens for use with other services.
- **Cloud Storage** - store and serve user-generated content like save games, images, audio, video or binary data.
- **Remote Config** - update your game without deploying a new version and customize content for different Firebase Analytics audiences and measure results.
- **Performance Monitoring** – get insights into how your game performs from your users' point of view, with automatic and customized performance tracing.
- **Crashlytics** - track, prioritize, and fix stability issues with lightweight but powerful, realtime crash reporter that improves your game quality.
- **Dynamic Links** - improve acquisition and engagement by bringing users directly to content that they were originally searching for, whether they have your game installed or not.
- **Invites** - complete solution for game referrals and sharing, free email and SMS delivery, let your existing players easy share your game.
- **In-App Messaging** - engage users by sending them targeted and contextual messages that nudge them to complete key in-app actions.

- **Test Lab** - test your games on physical and virtual devices hosted by Google that allow you to run tests that simulate actual usage environments.
- **Cloud Functions (server side)** - run backend code without managing servers and keep your logic private and secure.
- **Predictions** - use the power of Google's machine learning to create dynamic user groups based on players' predicted behavior.
- **A/B Testing** - create experiments to optimize the users experience for a business goal.
- **Hosting** - deploy web page with speed and security without all the hassle.
- All features are exposed to Blueprints.
- Supports gameDNA installer. No more downloading SDKs and creating packages on your own!
- Out-of-the-box for mobile platforms: iOS & Android.
- Works with Blueprint-only & source code projects.
- Works with Launcher & GitHub UE4 versions.

3. Plugin updates

a. Update from previous versions to 1.12.0

Delete all files under *Plugins/UltimateMobileKit* directory and install plugin again as described in *Getting started* section.

4. Example project

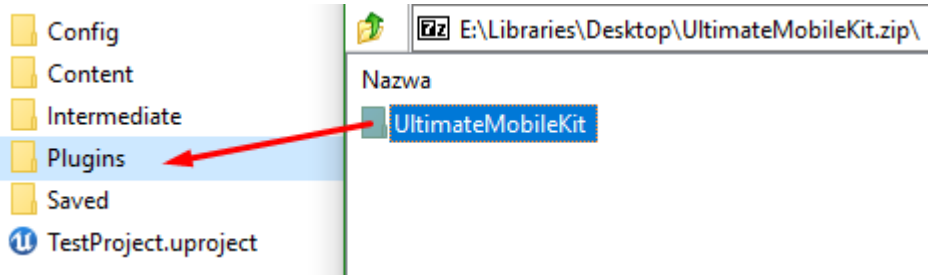
You can download an example project at the following address:
https://github.com/gameDNAstudio/ExampleProjects

5. GitHub Repository

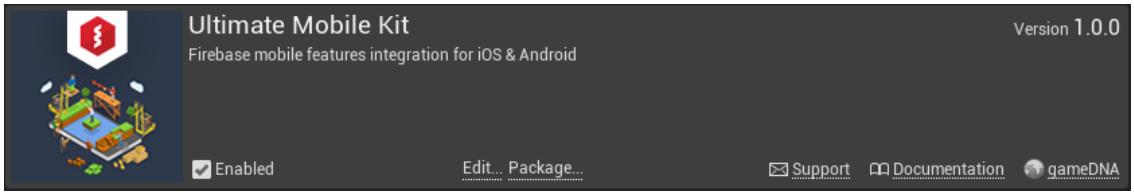
Please email the receipt of this plugin's purchase to support@gamednastudio.com for access to the GitHub repository used to develop this plugin!

6. Getting started

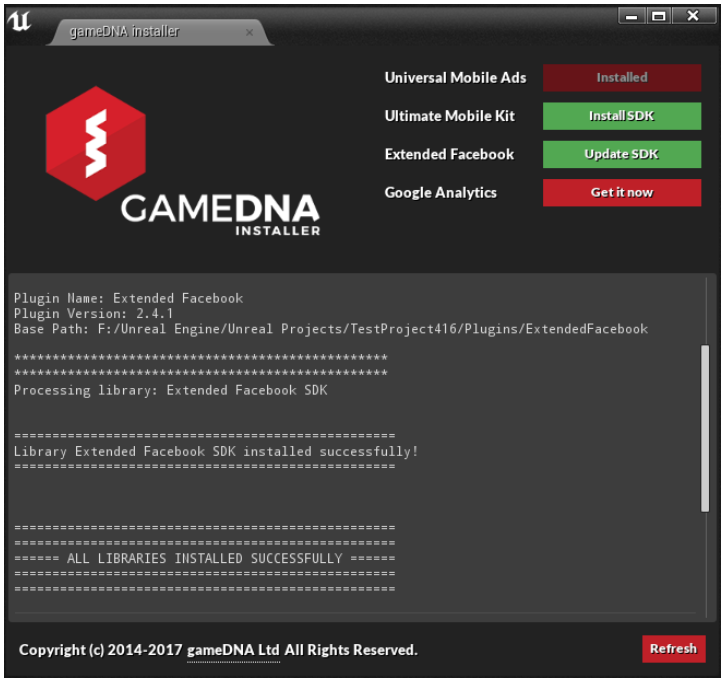
1. Go to https://firebase.google.com, create a Firebase Account and setup your game.
2. Unpack the plugin archive to *Plugins* folder in your UE4 project folder (for project plugins) or *Engine/Plugins/Marketplace* (for engine plugins) and start the editor.



3. Enable *Ultimate Mobile Kit* in *Edit -> Plugins -> Mobile -> Ultimate Mobile Kit*.



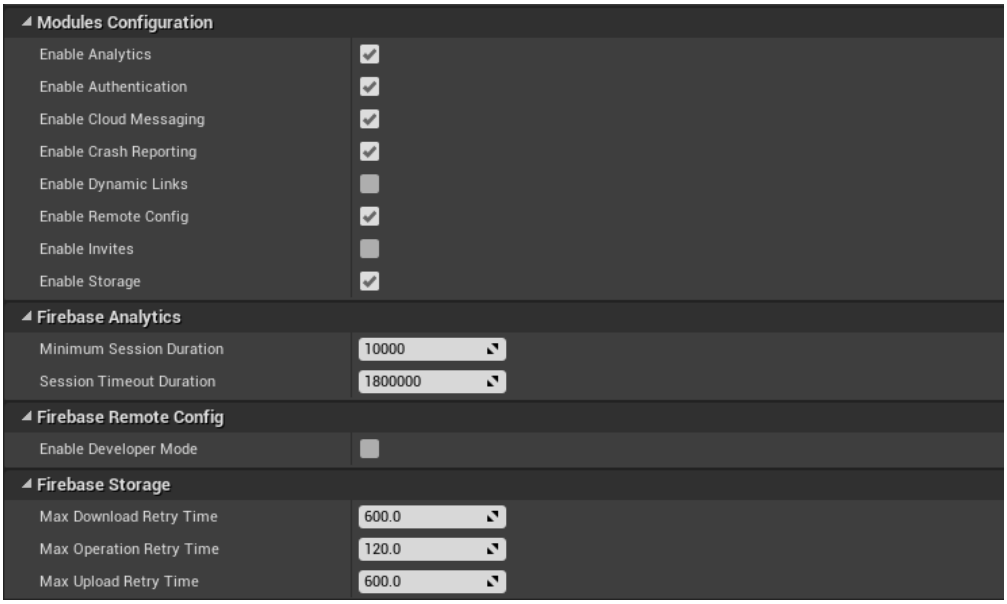
4. Download **gameDNA installer** plugin from the *releases* tab on the official GitHub repository https://github.com/gameDNAstudio/gameDNAinstaller. Install plugin and SDKs as described in the included documentation.



5. Android configuration:
   a) Download and copy configuration files to the following folder: *[PROJECT_FOLDER]/Config/Firebase/Android/google-services.json*
   b) Click on *Load Firebase Config* in the Android section.

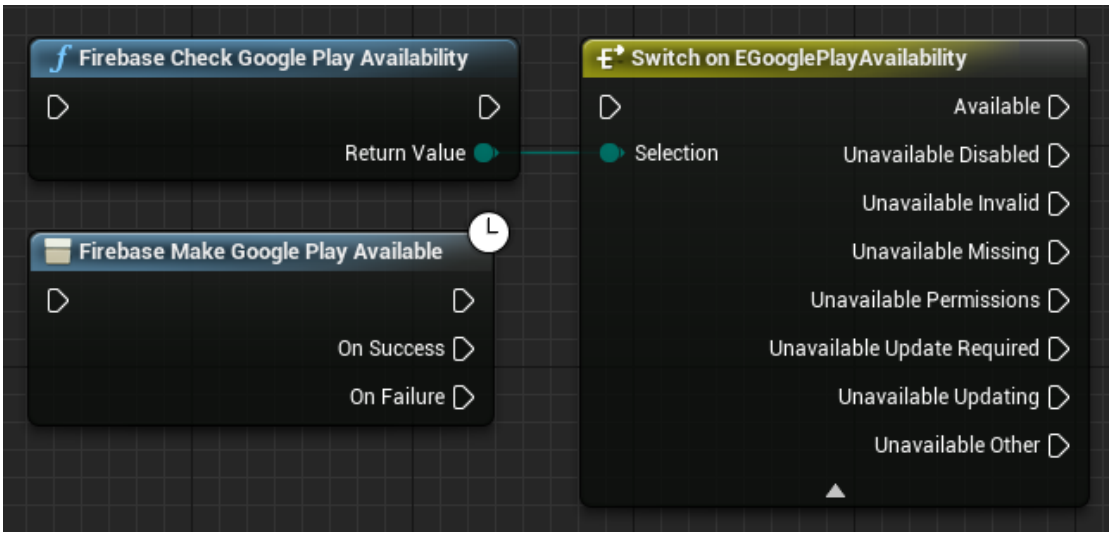
6. iOS configuration:
   a) Download and copy configuration files to the following folder: *[PROJECT_FOLDER]/Config/Firebase/IOS/GoogleService-Info.plist*
   b) Click on *Load Firebase Config* in the iOS section.

7. Go to *Project Settings -> Plugins -> Ultimate Mobile Kit*, select Firebase modules that should be enabled, and customize options.



8. Google Play Services

Many Firebase libraries require Google Play Services on the user's Android device. Before initializing any Firebase module, you should call the *Firebase Check Google Play Availability* function, which returns status of the Google Play Services. If it returns *Unavailable* state, it means Google Play Services need to be updated, reactivated, permissions fixed, etc. Call *Firebase Make Google Play Available* to fix this. If it's still not available, Firebase won't work and your game should handle this case.


Firebase Check Google Play Availability
Return Value
Firebase Make Google Play Available
On Success
On Failure
Switch on EGooglePlayAvailability
Selection
Available
Unavailable Disabled
Unavailable Invalid
Unavailable Missing
Unavailable Permissions
Unavailable Update Required
Unavailable Updating
Unavailable Other

7. Mobile Analytics

At the heart of Firebase is Firebase Analytics, an unlimited analytics solution. Analytics integrates across Firebase features and provides you with unlimited reporting for up to 500 distinct events that you can define using the Ultimate Mobile Kit plugin. Firebase Analytics reports help you understand clearly how your users behave, which enables you to make informed decisions regarding app marketing and performance optimizations.

Firebase Analytics helps you understand how people use your iOS or Android game. The SDK automatically captures a number of events and user properties. It also allows you to define your own custom events to measure the things that uniquely matter to your business. Once the data is captured, it's available in a dashboard through the Firebase console. This dashboard provides detailed insights about your data — from summary data such as active users and demographics, to more detailed data such as identifying your most purchased items.

Firebase Analytics also integrates with a number of other Firebase features. For example, it automatically logs events that correspond to your Firebase Notifications and provides reporting on the impact of each campaign.

See the performance of your campaigns across organic and paid channels to understand which methods are most effective at driving high-value users. If you need to perform custom analysis or join your data with other sources, you can link your Analytics data to BigQuery, which allows for more complex analysis like querying large data sets and joining multiple data sources.

Key capabilities:

- **Unlimited Reporting** - Firebase Analytics provides unlimited reporting on up to 500 distinct events.
- **Audience Segmentation** - custom audiences can be defined in the Firebase console based on device data, custom events, or user properties. These audiences can be used with other Firebase features when targeting new features or notifications.

Integrations with other services:

- **BigQuery** - link your Firebase Analytics game to BigQuery where you can perform custom analysis on your entire Analytics dataset and import other data sources.

- **Firebase Crash Reporting** - Firebase Analytics logs events for each crash so you can get a sense of the rate of crashes for different versions or regions, allowing you to gain insight into which users are impacted. You can also create audiences for users who have experienced multiple crashes and respond with Firebase Notifications directed at that audience.
- **Firebase Notifications** - Firebase Analytics automatically logs events that correspond to your Firebase Notifications and supports reporting on the impact of each campaign.
- **Firebase Remote Config** - use Firebase Analytics audience definitions to change the behavior and appearance of your app for different audiences without distributing multiple versions of your app.
- **Google Tag Manager** - integrating Google Tag Manager alongside Firebase Analytics enables you to manage your Firebase Analytics implementation remotely from a web interface after your app has been distributed.

Official Firebase Analytics documentation:
https://firebase.google.com/docs/analytics/

Video introduction to Firebase Analytics: https://youtu.be/iT6EaIwtonY

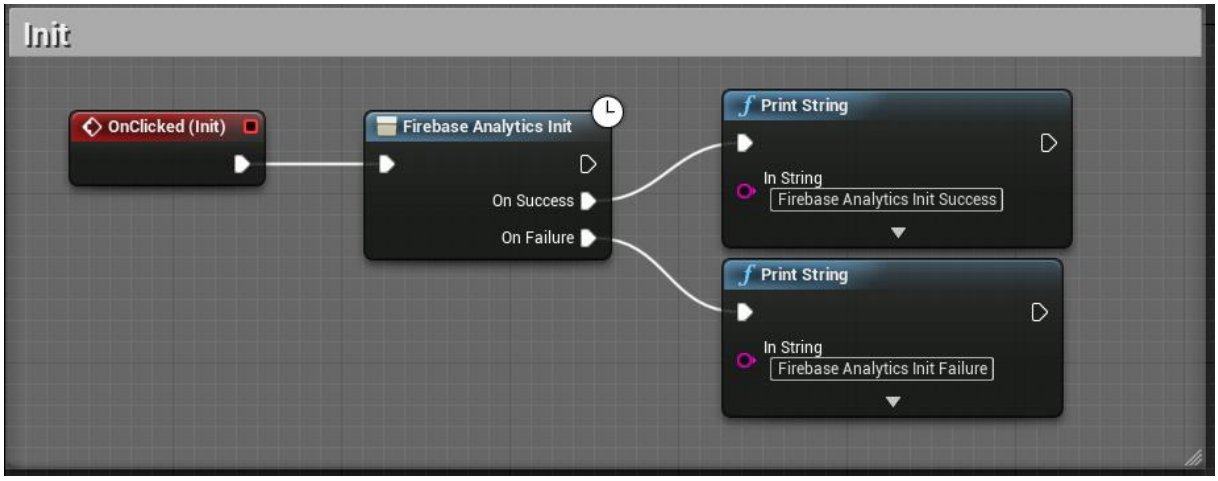
Firebase Analytics collects usage and behavior data for your game. The SDK logs two primary types of information:

- **Events**: What is happening in your app, such as user actions, system events, or errors.
- **User properties**: Attributes you define to describe segments of your userbase, such as language preference or geographic location.

Note: Data in the Analytics reporting dashboard refreshes periodically throughout the day.

a. Initialization

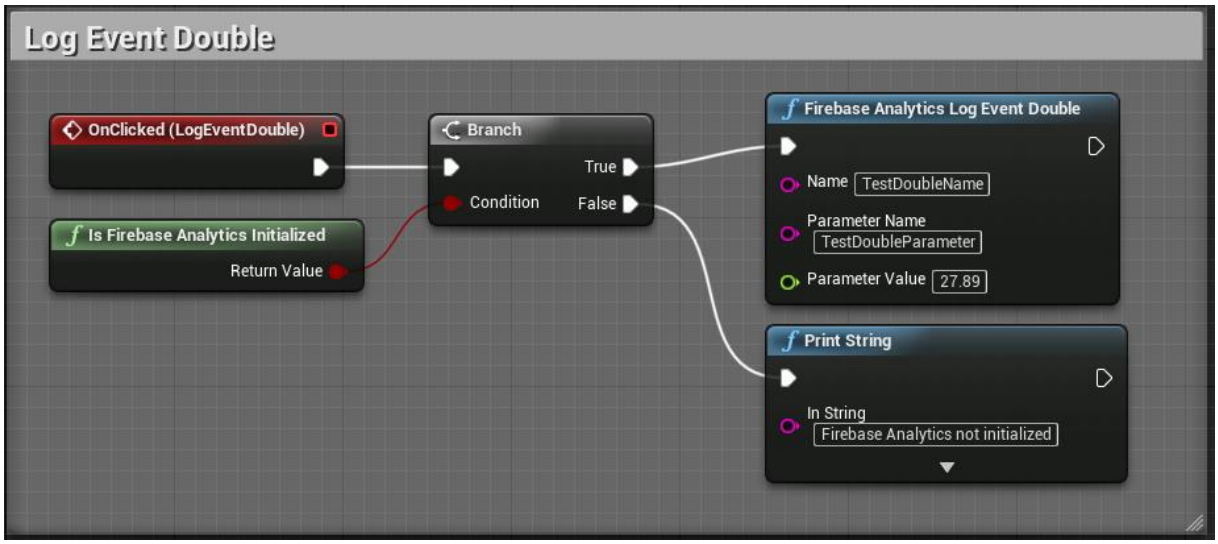
Before making any call to the Firebase Analytics, you should first execute the *Firebase Analytics Init* function. If the result is a fail, study logs to find out what causes an issue.

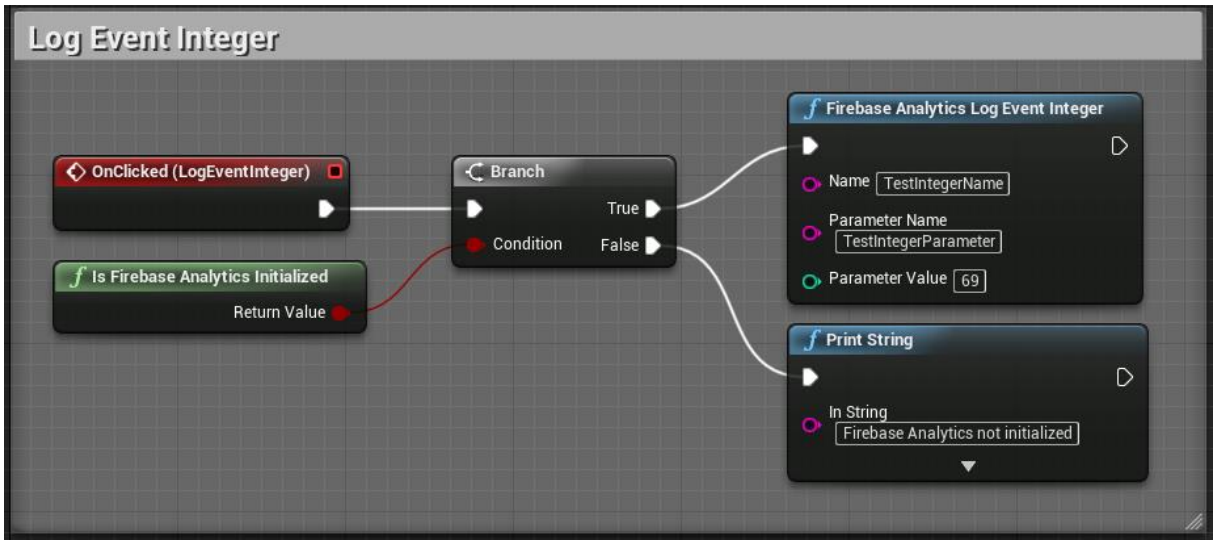


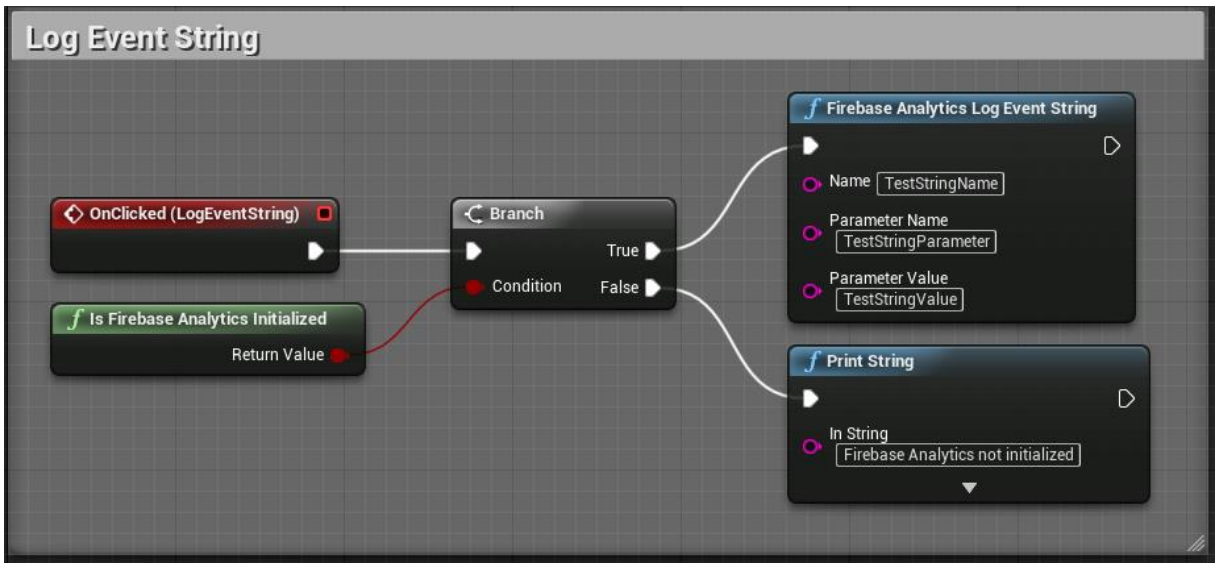
b. Log Events

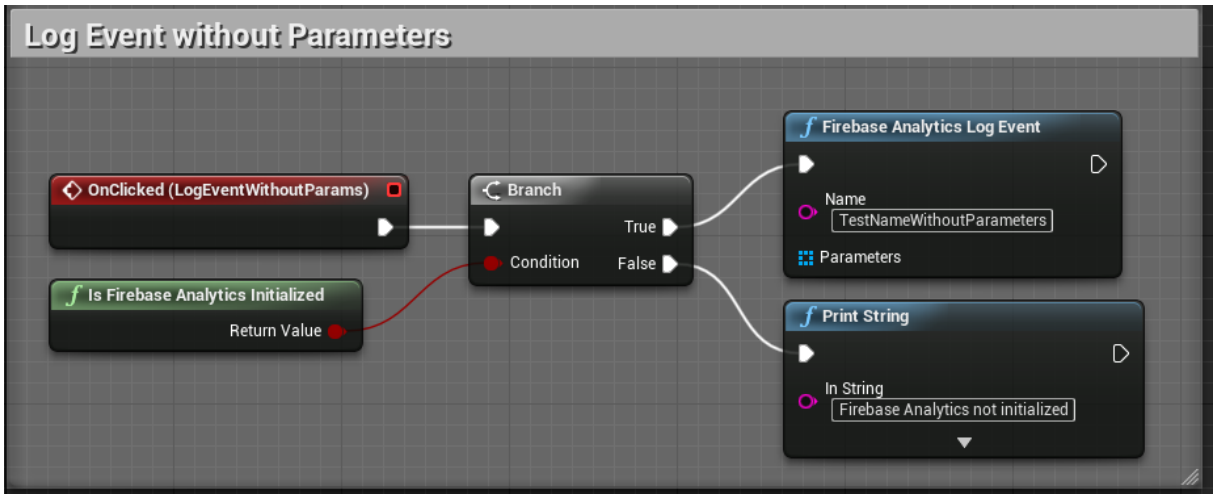
After you have configured a Firebase Analytics instance, you can begin to log events. Events provide insight on what is happening in your game, such as user actions, system events, or errors. Analytics automatically logs some events and user properties; you don't need to add any code to enable them. If your game needs to collect additional data, you can log up to 500 different Analytics Event types in your game. There is no limit on the total volume of events your game logs.

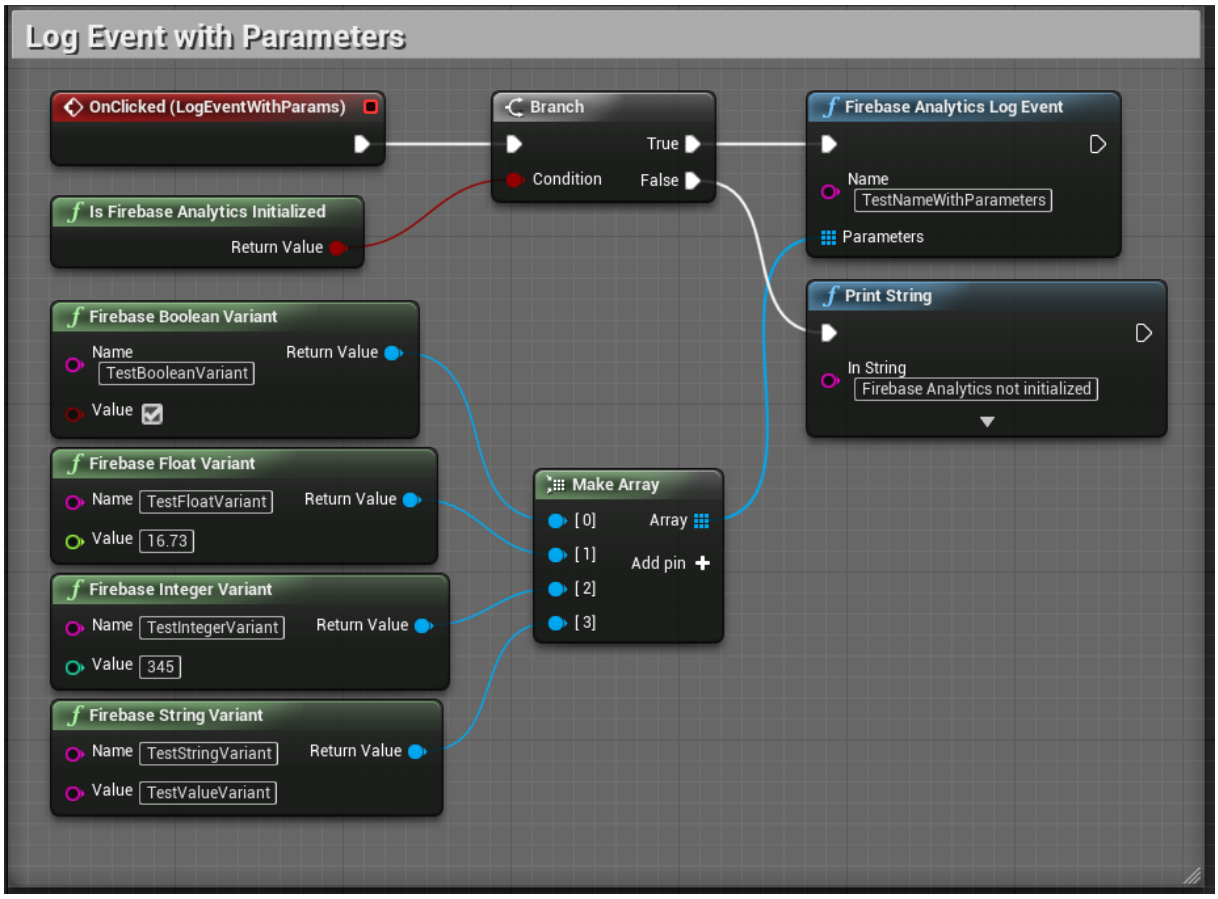
You can log events without parameters, events with a string, double and integer types of parameters, and events with multiple parameters (using *Firebase Variant*).




Log Event Integer
OnClicked (LogEventInteger)
Branch
True
Condition
False
Is Firebase Analytics Initialized
Return Value
Firebase Analytics Log Event Integer
Name
TestIntegerName
Parameter Name
TestIntegerParameter
Parameter Value
69
Print String
In String
Firebase Analytics not initialized


Log Event String
OnClicked (LogEventString)
Branch
True
Condition
False
Is Firebase Analytics Initialized
Return Value
Firebase Analytics Log Event String
Name
TestStringName
Parameter Name
TestStringParameter
Parameter Value
TestStringValue
Print String
In String
Firebase Analytics not initialized

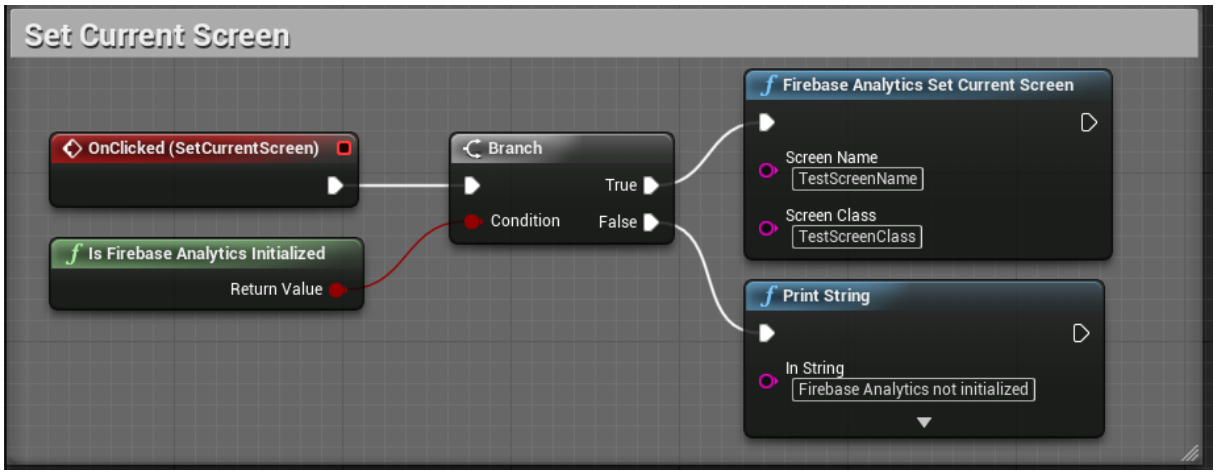

Log Event without Parameters
OnClicked (LogEventWithoutParams)
Branch
True
Condition
False
Is Firebase Analytics Initialized
Return Value
Firebase Analytics Log Event
Name
TestNameWithoutParameters
Parameters
Print String
In String
Firebase Analytics not initialized



c. Set Current Screen

Sets the current screen name and screen class, which specifies the current visual context in your game.

This helps identify the areas in your game where users spend their time and how they interact with your game.

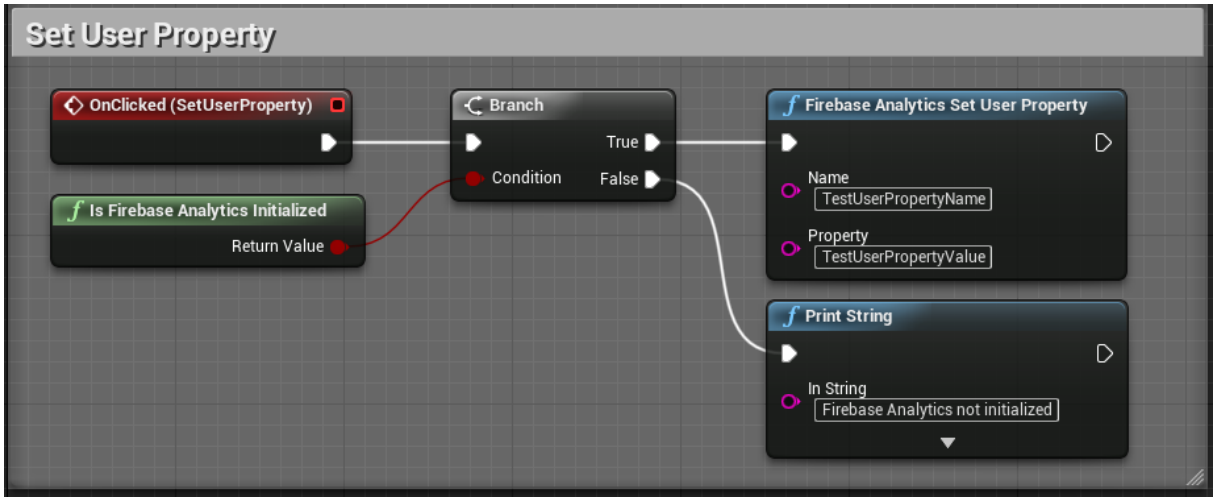


d. Set User Property

User properties are attributes you define to describe segments of your userbase, such as language preference or geographic location.

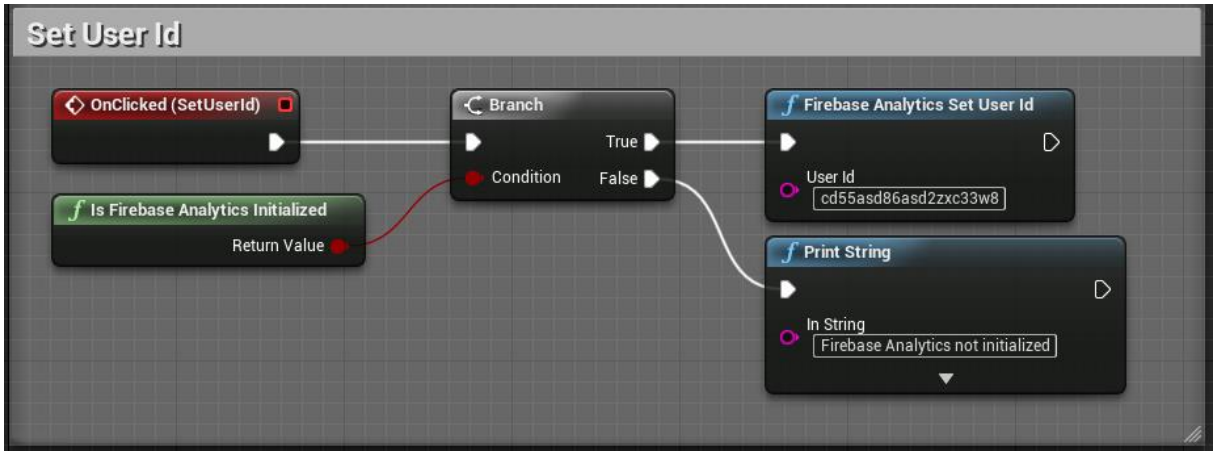
Analytics automatically logs some user properties; you don't need to add any code to enable them. If your game needs to collect additional data, you can set up to 25 different Analytics User Properties in your game.

You can set Analytics user properties to describe the users of your game. You can analyze behaviors of various user segments by applying these properties as filters to your reports. Property must be registered on the Analytics page of the Firebase console.



e. Set User ID

Firebase automatically generates a user ID for the player in your game but you can assign a custom user ID. This feature must be used in accordance with Google's Privacy Policy.



8. Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost.

Using FCM, you can notify a client app that a new event or other data is available to sync. You can send notification messages to drive user reengagement and retention. For use cases, such as instant messaging, a message can transfer a payload of up to 4KB to a client game.

Key capabilities:

- **Send notification messages or data messages** - send notification messages that are displayed to your user. Or send data messages and determine completely what happens in your game code.
- **Versatile message targeting** - distribute messages to your client game in any of three ways — to single devices, to groups of devices, or to devices subscribed to topics.
- **Send messages from client apps** - send acknowledgments, chats, and other messages from devices back to your server over FCM's reliable and battery-efficient connection channel.

An FCM implementation includes two main components for sending and receiving:

- A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target and send messages.
- An iOS or Android client game that receives messages.

You can send messages via the HTTP and XMPP APIs. For testing or for sending marketing or engagement messages with powerful built-in targeting and analytics, you can also use the Notifications composer.

Official Firebase Cloud Messaging documentation:
https://firebase.google.com/docs/cloud-messaging/

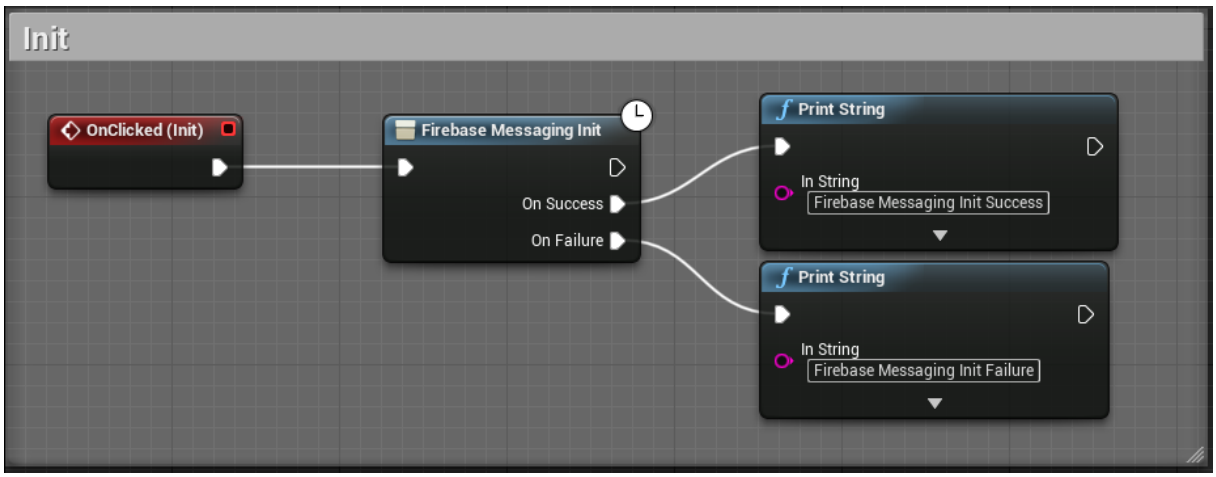
About FCM Messages: https://firebase.google.com/docs/cloud-messaging/concept-options

Video introduction to Firebase Cloud Messaging: https://youtu.be/sioEY4tWmLI

Only for iOS: You need a valid APNs certificate. If you don't already have one, see Provision APNs SSL Certificates.

a. Initialization

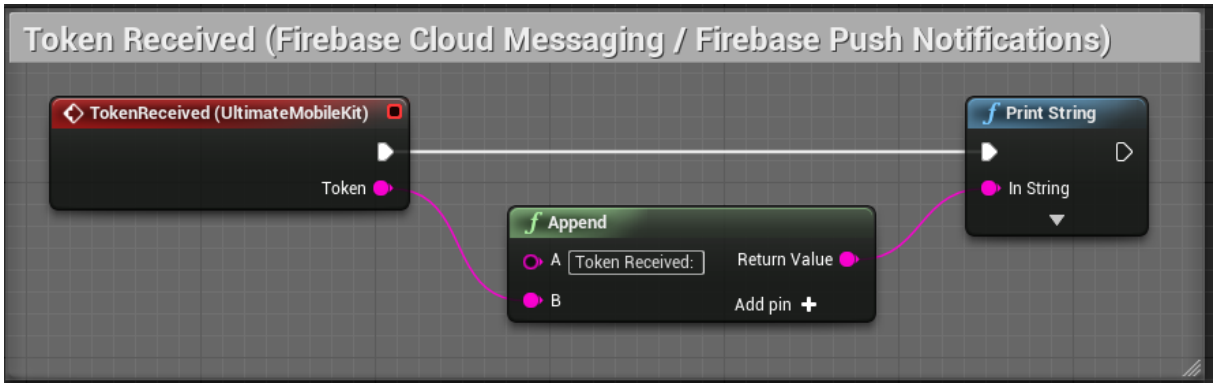
Before making any call to Firebase Cloud Messaging, you should first execute the *Firebase Messaging Init* function. If the result is a fail, study logs to find out what causes an issue.



b. Access the device registration token

On initial startup of your game, the FCM SDK generates a registration token for the client game instance. If you want to target single devices, or create device groups for FCM, you'll need to access this token.

You can access the token's value through the *TokenReceived* callback in the *UltimateMobileKit* component.



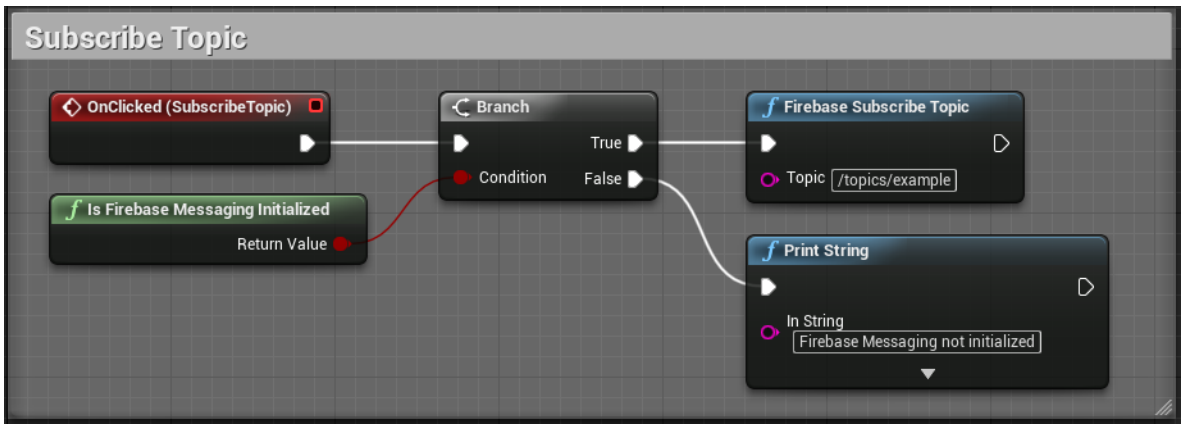
c. Subscribe/Unsubscribe

Based on the publish/subscribe model, FCM topic messaging allows you to send a message to multiple devices that have opted in to a particular topic. You compose topic messages as needed, and FCM handles routing and delivering the message reliably to the right devices.

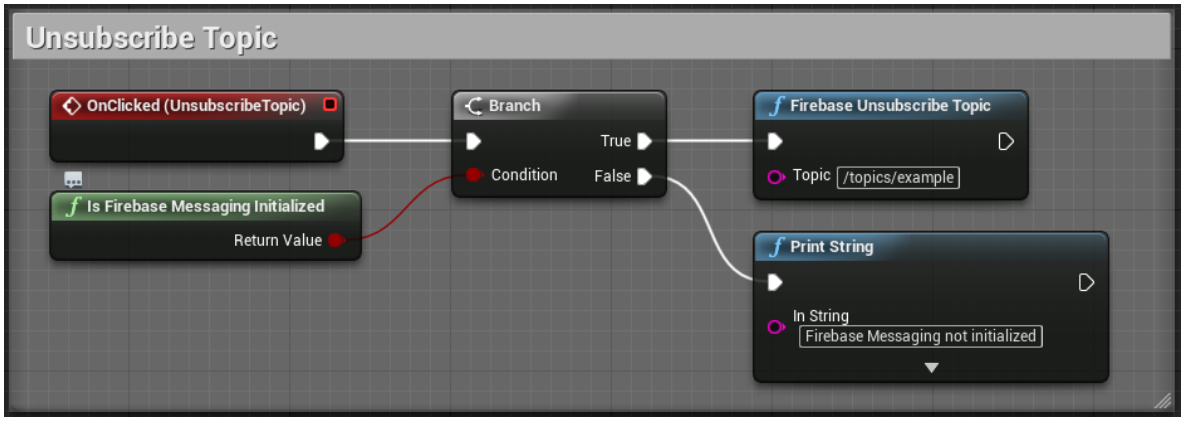
Some things to keep in mind about topics:

- Topic messaging supports unlimited topics and subscriptions for each app.
- Topic messages are optimized for throughput rather than latency. For fast, secure delivery to single devices or small groups of devices, target messages to registration tokens, not topics.
- If you need to send messages to multiple devices per user, consider device group messaging for those use cases.

To subscribe to a topic, call *Firebase Subscribe Topic* from your game. This makes an asynchronous request to the FCM backend and subscribes the client to the given topic. If the subscription request fails initially, FCM retries until it can subscribe to the topic successfully. Each time the app starts, FCM makes sure that all requested topics have been subscribed.



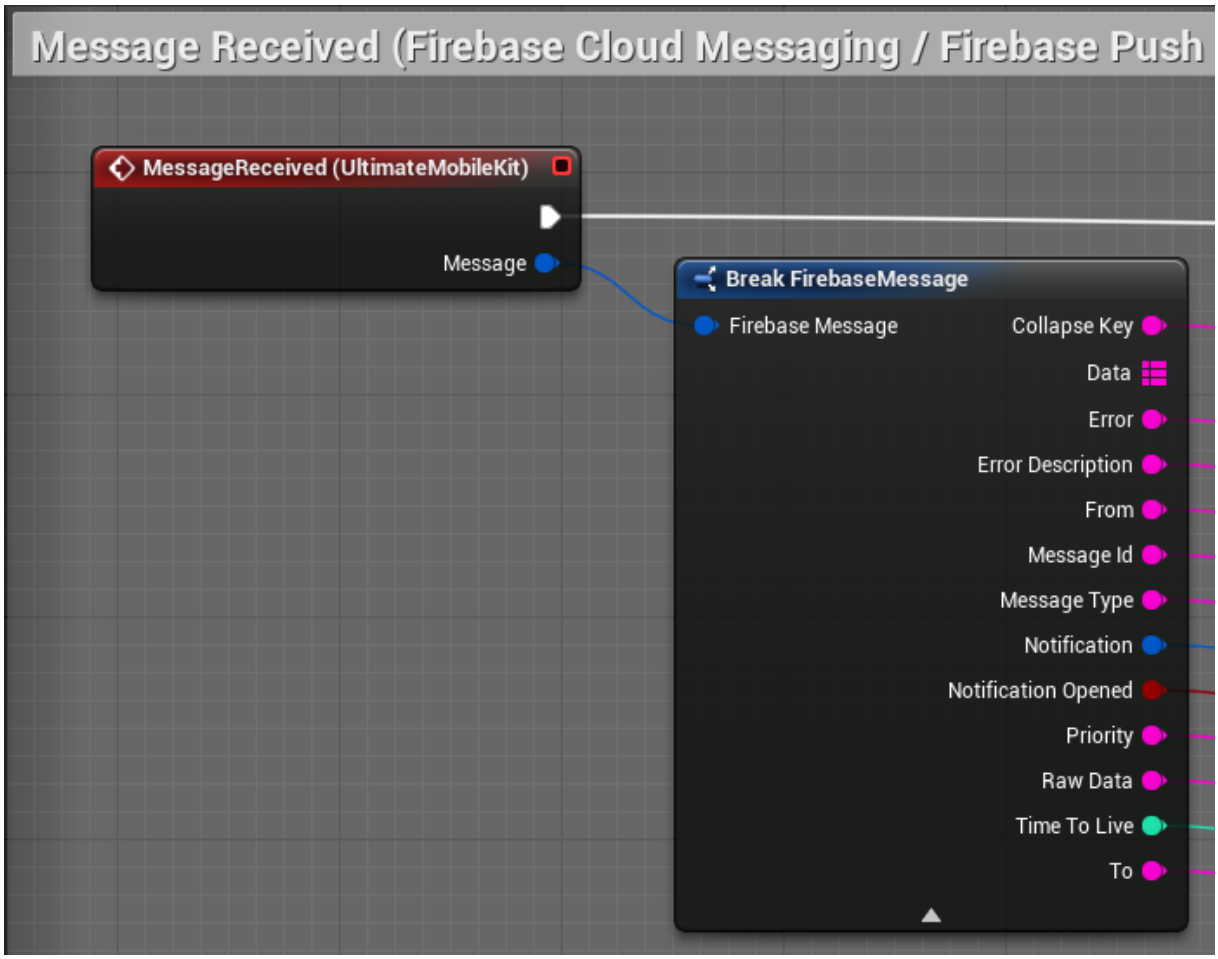
To unsubscribe, call *Firebase Unsubscribe Topic*, and FCM unsubscribes from the topic in the background.



d. Receive messages

FCM delivers topic messages in the same way as other downstream messages.

By handling the *MessageReceived* callback in the *UltimateMobileKit* component, you can perform actions based on the received message and get the message data.



e. Send downstream messages

You can send downstream messages to a device by making HTTP or XMPP requests. You will find more about it on the official Firebase documentation:

- Send Messages to Topic
- Send Messages to Device Groups

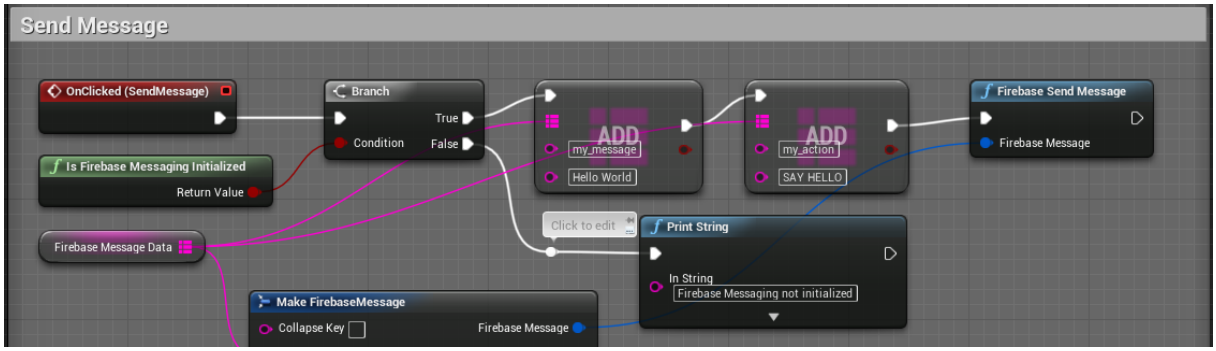
f. Send upstream messages

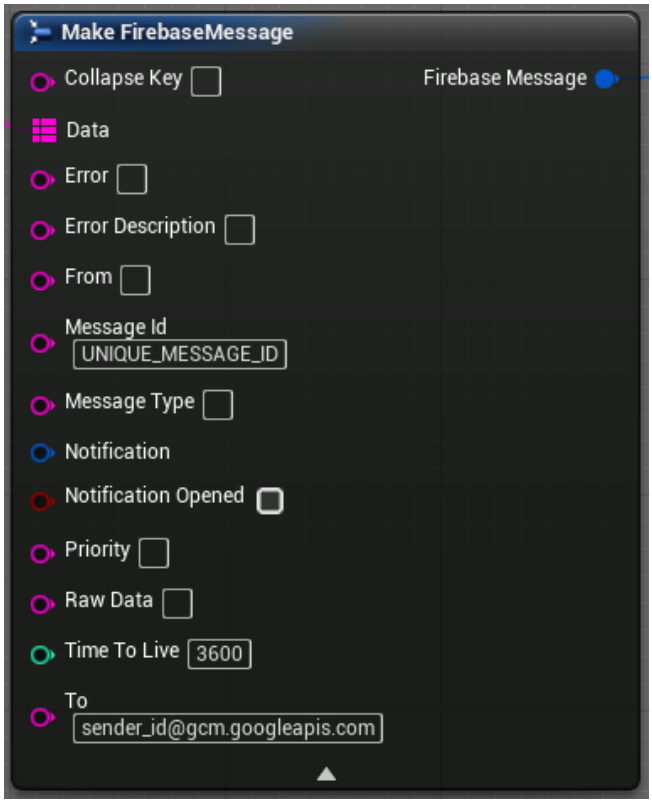
If your game server implements the XMPP Connection Server protocol, it can receive upstream messages from a user's device to the cloud. To initiate an upstream message, the client game sends a request containing the following:

- The address of the receiving game server in the format SENDER_ID@gcm.googleapis.com.
- A message ID that should be unique for each sender ID.
- The message data comprising the key-value pairs of the message's payload.

When it receives this data, FCM builds an XMPP stanza to send to the game server, adding some additional information about the sending device and game.

More info on the official Firebase documentation.





9. Push Notifications

Firebase Notifications is a service that enables targeted user notifications for mobile game developers.

Built on Firebase Cloud Messaging (FCM) and the FCM SDK, Firebase Notifications provides an option for developers and organizations seeking a flexible notification platform that requires minimal coding effort to get started, and a graphical console for sending messages. Using the Notifications console GUI, you can reengage and retain your userbase, faster app growth, and support marketing campaigns.

Notifications integrates closely with Firebase Analytics, allowing you to target notifications by custom audience. Also, you can target predefined user segments for game, version, and language. With notifications targeted to user segments, you can contact exactly the right user audience for updates on available upgrades, new features, or other news.

Use the Notifications console GUI to compose and send notifications to all supported message targets. Firebase Cloud Messaging handles the routing and delivery to targeted devices.

When your game is in the background on a user's device, notifications are delivered to the system tray. When a user taps on the notification, the game launcher opens your game. If you want, you can also add client message handling to receive notifications in your game when it is already in the foreground on the user's device.

Key capabilities:

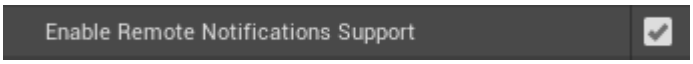
- **Notifications analytics** - analyze reengagement conversion with built-in notifications analytics collection and funnel analysis.
- **Versatile message targeting** - target clients in predefined user segments, custom analytics audiences, clients subscribed to topics, and single devices.
- **Flexible message scheduling** - deliver notifications (up to 2kb) immediately, or at a future time in the client's local time.

Official Firebase Push Notifications documentation:
https://firebase.google.com/docs/notifications/

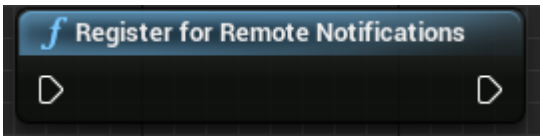
Video introduction to Firebase Push Notifications: https://youtu.be/rTB7fTqMlS0

a. Enable Push Notifications

1. Go to *Project Settings -> IOS -> Enable Remote Notifications Support* and check whether the option is enabled.



2. Call the *Register for Remote Notifications* function before the Firebase Messaging initialization call.



3. **Only for iOS:** You need a valid APNs certificate. If you don't already have one, see Provision APNs SSL Certificates.
4. **Only for Android:** You can change default notifications icon under the path *[PROJECT_FOLDER]/Source/ThirdParty/AndroidAddons/res/drawable-[*]/ic_stat_ic_notification.png*
5. Other steps are exactly the same as in the case of Cloud Messaging.

b. Send a message from the Notifications console

1. Open the Notifications tab of the Firebase console and select *New Message*.
2. Enter a message text.
3. Send a Notification:
   a. **[Send to a user segment]** Select the message target. The dialog displays further options to refine the target based on whether you choose *App/App Version*, *Device Language*, *Users in Audience* or *User Properties*. After you click *Send Message*, targeted client devices that have the app in the background receive the notification in the system notifications tray. When a user taps on the notification, the app launcher opens your app.
   b. **[Send to a topic]** Select *Topic* for the message target. From the topics list, select the topic you want to target. This list is populated with all topics with active subscriptions in this Firebase project (there may be a delay of as much as one day before new topics are displayed in the console). After you click *Send Message*, Firebase delivers the message to all client app instances that are subscribed to the topic.
4. By handling the *MessageReceived* callback in the *UltimateMobileKit* component, you can perform actions based on the received notification and get the notification data (like in Cloud Messaging).

10. Authentication

Most games need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use functions, and ready-made UI libraries to authenticate users to your game. It supports authentication using passwords, popular federated identity providers like Google, Facebook and Twitter, and more.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

Firebase Authentication is compatible with another gameDNA's plugin, Extended Facebook Online Subsystem.

To sign a user into your app, you first get authentication credentials from the user. These credentials can be the user's email address and password, or an OAuth token from a federated identity provider. Then, you pass these credentials to the Firebase Authentication SDK. Google's backend services will then verify those credentials and return a response to the client.

After a successful sign in, you can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products. You can also use the provided authentication token to verify the identity of users in your own backend services.

Key capabilities:

- **Email and password based authentication** - authenticate users with their email addresses and passwords. Ultimate Mobile Kit provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails.
- **Federated identity provider integration -** authenticate users by integrating with federated identity providers. The Firebase Authentication SDK provides methods that allow users to sign in with their Google, Google Play, Facebook, Game Center, Twitter, and GitHub accounts.
- **Phone number authentication -** authenticate users by sending SMS messages to their phones.

- **Custom auth system integration** - connect your app's existing sign-in system to the Firebase Authentication SDK and gain access to Firebase Realtime Database and other Firebase services.
- **Anonymous auth** - use Firebase features that require authentication without requiring users to sign in first by creating temporary anonymous accounts. If the user later chooses to sign up, you can upgrade the anonymous account to a regular account, so the user can continue where they left off.

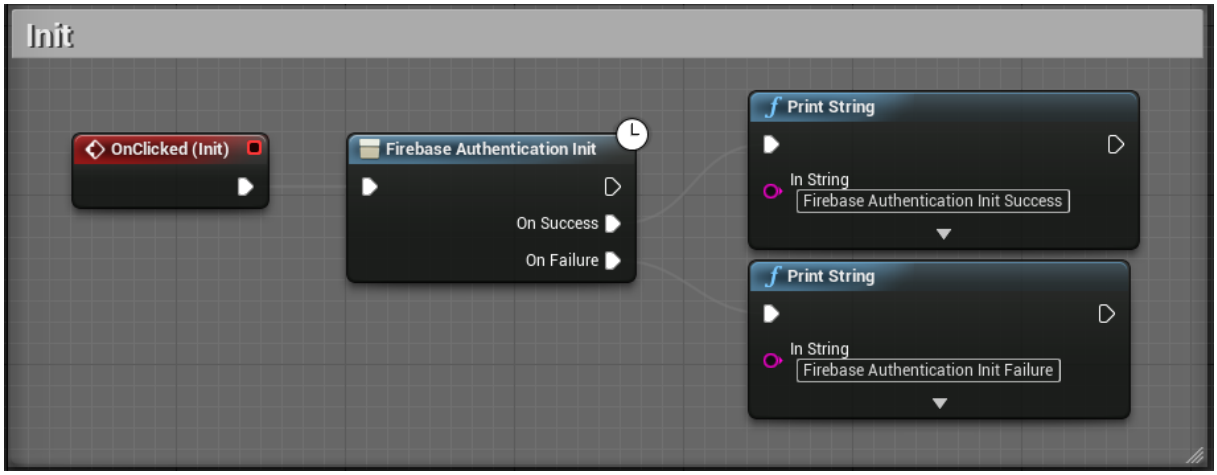
Official Firebase Authentication documentation:
https://firebase.google.com/docs/auth/

Users in Firebase Projects: https://firebase.google.com/docs/auth/users

Video introduction to Firebase Authentication: https://youtu.be/8sGY55yxicA

a. Initialization

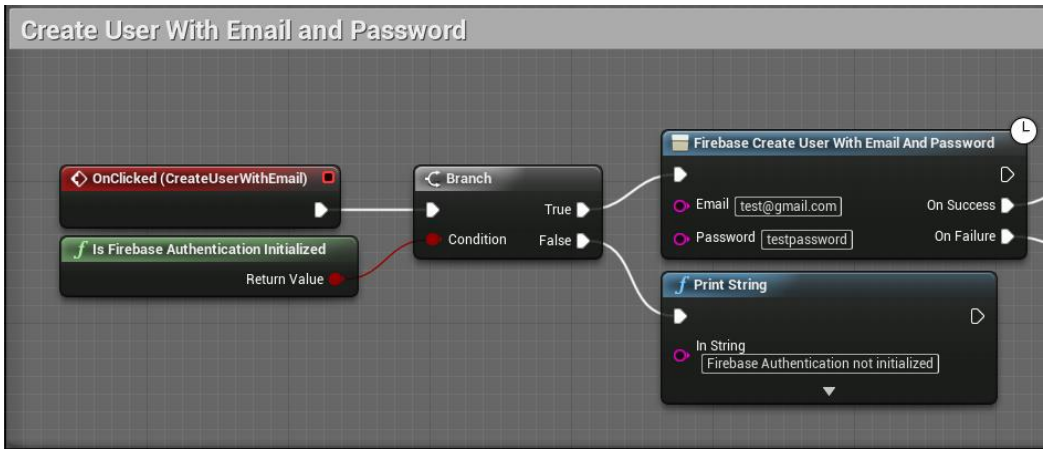
Before making any call to Authentication, you should first execute *Firebase Authentication Init* function. If the result is a fail, study logs to find out what causes an issue.



b. Create User

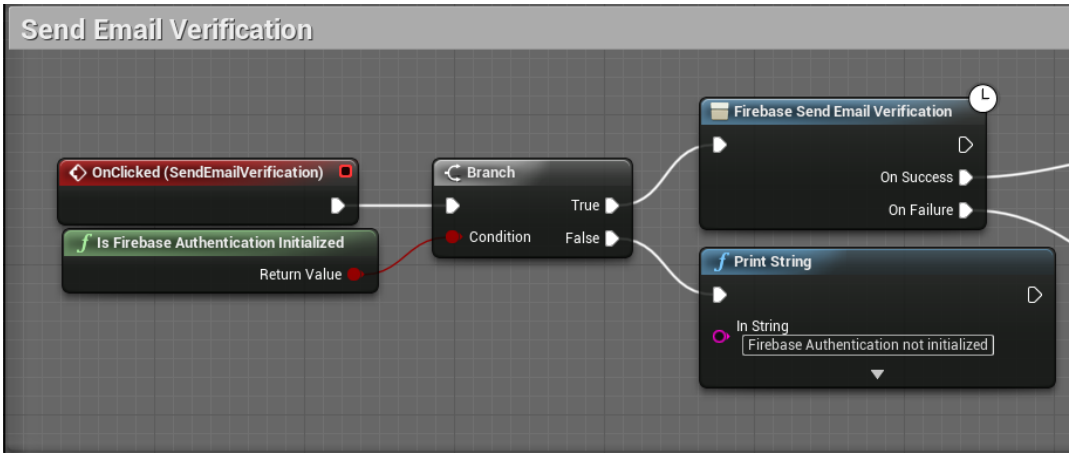
i. Create User With Email And Password

Create a form that allows new users to register with your app using their email address and a password. When a user completes the form, validate the email address and password provided by the user, then pass them to the *Firebase Create User With Email And Password* function.



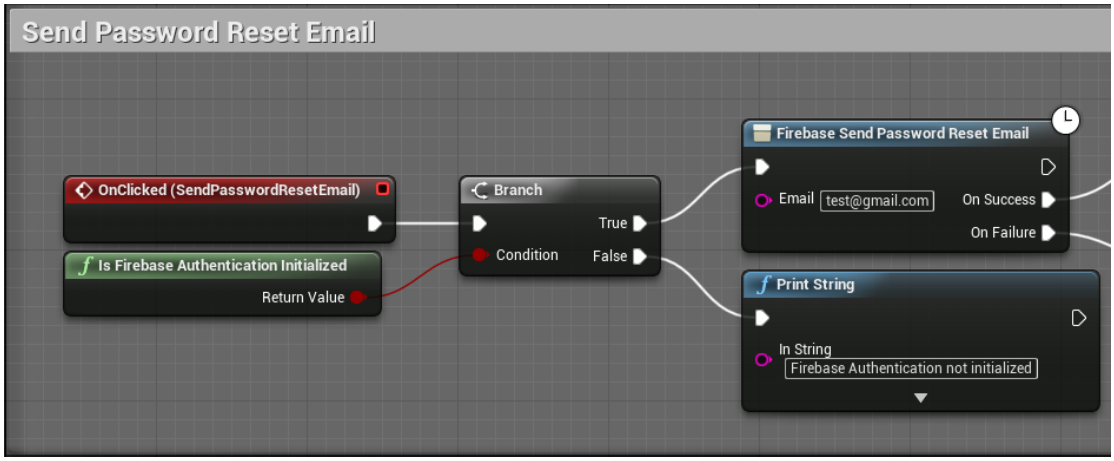
ii. Send Email Verification

You can send an address verification email to a user with the *Firebase Send Email Verification* function. You can customize the email template that is used in the Authentication section of the Firebase console, on the Email Templates page. See Email Templates in Firebase Help Center.



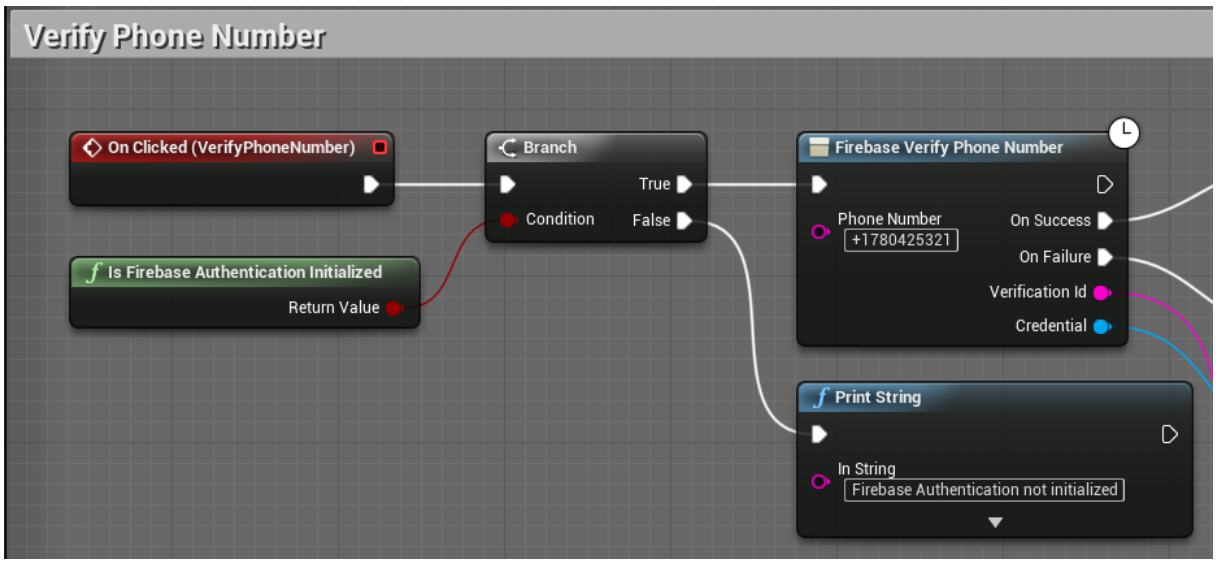
iii. Send Password Reset Email

You can send a password reset email to a user with the *Firebase Send Password Reset Email* method. You can customize the email template that is used in the Authentication section of the Firebase console, on the Email Templates page. See Email Templates in Firebase Help Center. You can also send password reset emails from the Firebase console.



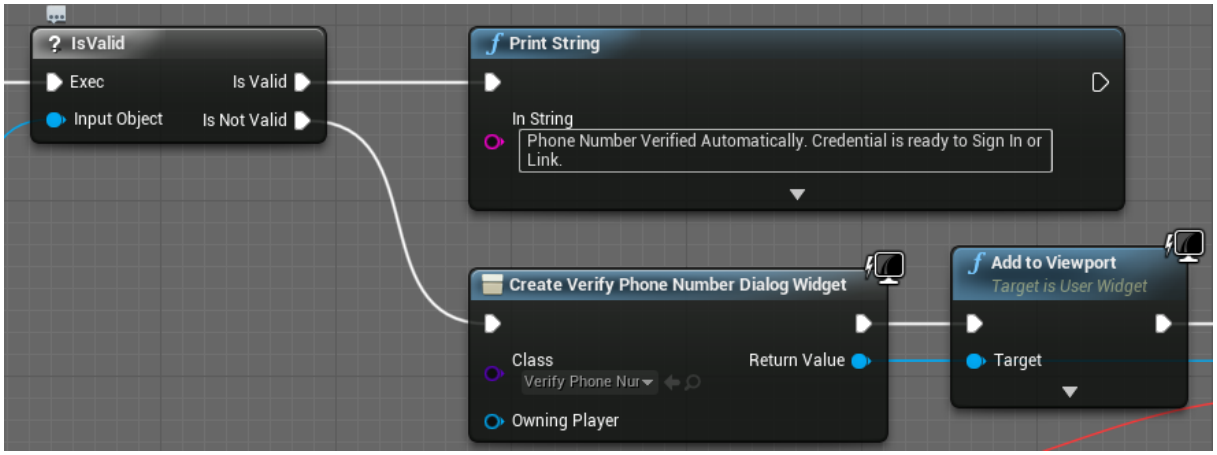
c. Verify Phone Number

You can use Firebase Authentication to sign in a user by sending an SMS message to the user's phone. The user signs in using a one-time code contained in the SMS message.



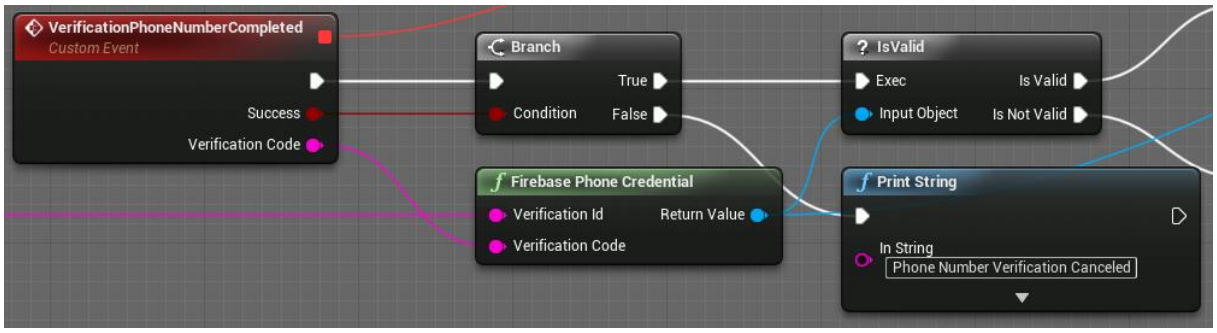
To initiate phone number sign-in, present the user an interface that prompts them to provide their phone number, and then call *Firebase Verify Phone Number* to request that Firebase send an authentication code to the user's phone by SMS:

1. Get the user's phone number.
   Legal requirements vary, but as a best practice and to set expectations for your users, you should inform them that if they use phone sign-in, they might receive an SMS message for verification and standard rates apply.
2. Call *Firebase Verify Phone Number*, passing to it the user's phone number.
   When you call *Firebase Verify Phone Number*, Firebase,
   - (on iOS) sends a silent push notification to your game,
   - sends an SMS message containing an authentication code to the specified phone number and passes a verification ID to your completion function. You will need both the verification code and the verification ID to sign in the user.
3. Save the verification ID and restore it when your game loads. By doing so, you can ensure that you still have a valid verification ID if your game is terminated before the user completes the sign-in flow (for example, while switching to the SMS app). You can persist the verification ID any way you want (for example in Save Games).



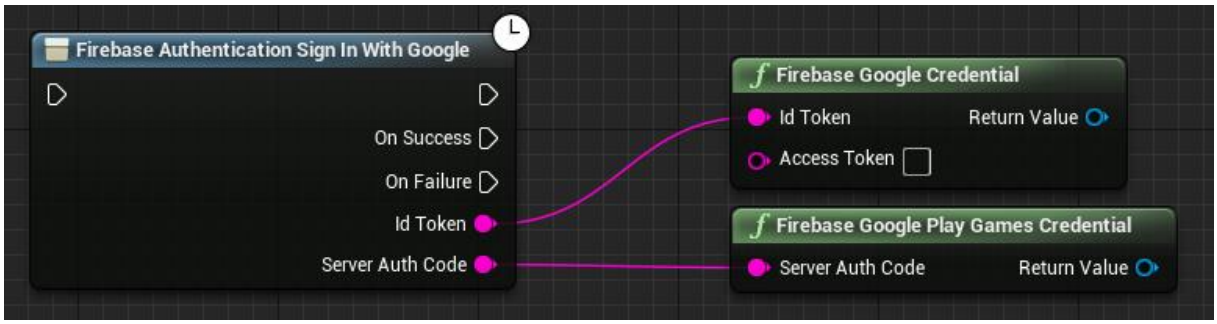
If the call to *Firebase Verify Phone Number* results in *Success* being called and *Credential* is not valid, you can prompt the user to type the verification code when they receive it in the SMS message.

On the other hand, if the call to *Firebase Verify Phone Number* results in *Success* and *Credential* is valid, then automatic verification has succeeded and you will now have a Credential which you can use to *Sign In With* Credential or *Link With Credential*.



d. Sign In With Google

You can call *Firebase Authentication Sign In With Google* function to login with *Google Account* and *Google Play Games Account*. After successful login you are able to obtain *Id Token* and *Server Auth Code* needed for *Firebase Google Credential* and *Firebase Google Play Games Credential*.



e. Link with external providers

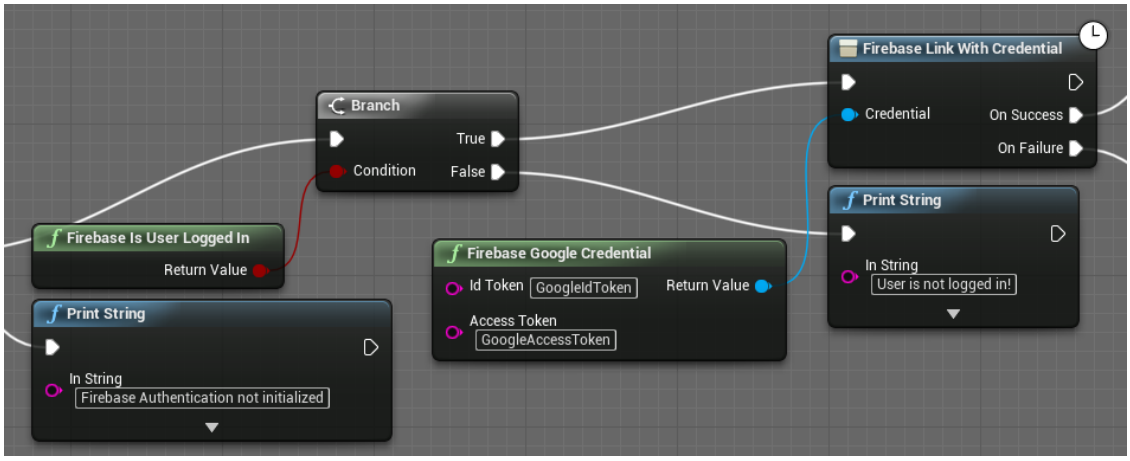
i. Link With Credential

You can allow users to sign in to your game using multiple authentication providers by linking auth provider credentials to an existing user account. Users are identifiable by the same Firebase user ID regardless of the authentication provider they used to sign in. For example, a user who signed in with a password can link a Google account and sign in with either method in the future. Or, an anonymous user can link a Facebook account and then, later, sign in with Facebook to continue using your game.

Currently supported providers:

- Email/Password (standard)
- Facebook
- Google
- Google Play Games
- Game Center
- Twitter
- GitHub
- Phone Number Authentication

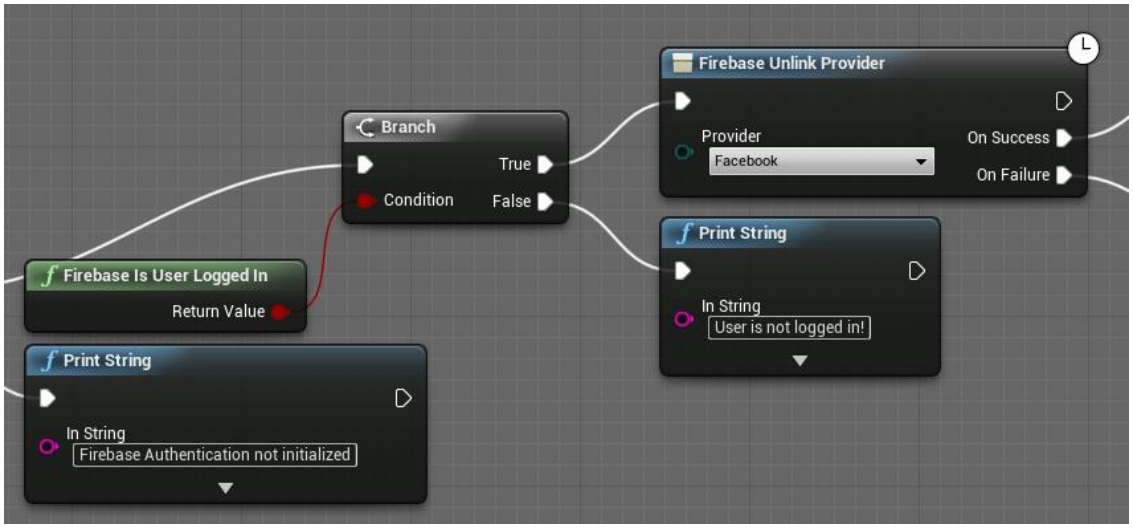
You can link provider with an existing user account by initializing appropriate *Firebase Credential* and calling *Firebase Link With Credential*. You should obtain tokens for Credential on your own (for example, for Facebook using [Extended Facebook Online Subsystem]).



If the call to *Firebase Link With Credential* succeeds, the user can now sign in using any linked authentication provider and access the same Firebase data.

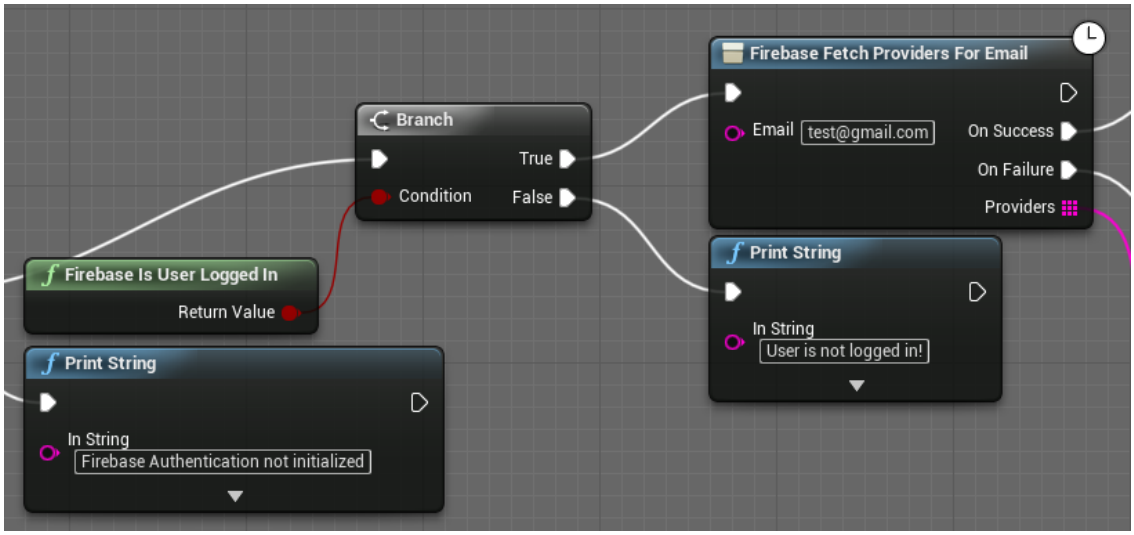
ii. Unlink Provider

You can unlink an auth provider from an account, so that the user can no longer sign in with that provider. To unlink an auth provider from a user account, pass the provider ID to the *Firebase Unlink Provider* function.



iii. Fetch Providers for Email

Firebase Fetch Providers for Email asynchronously requests the IDPs (identity providers) that can be used for the given email address. Useful for an "identifier-first" login flow.

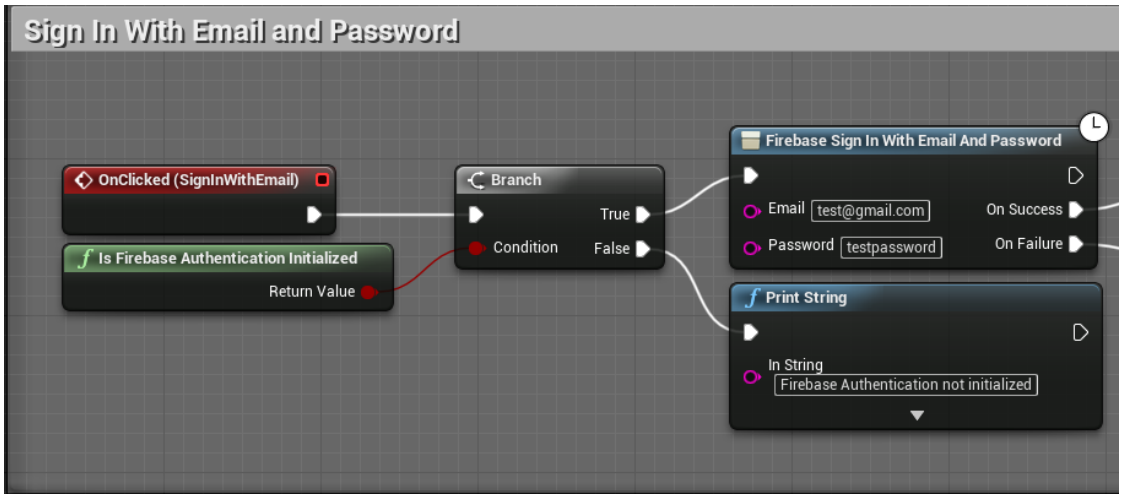


f. Sign In

Create a form that allows existing users to sign in using their email address and password. When a user completes the form, call the chosen signed in function.

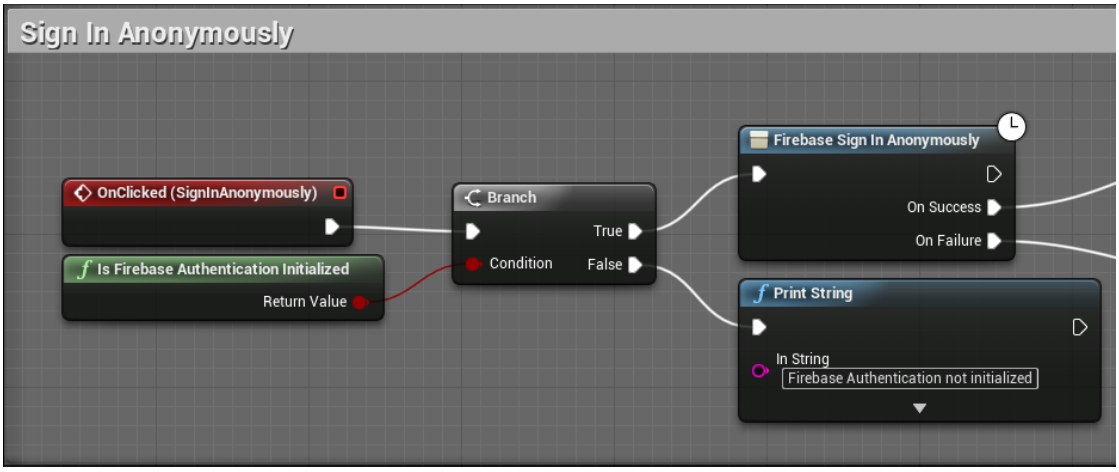
i. Sign In With Email And Password

The steps for signing in a user with a password are similar to the steps for creating a new account. When a user signs in to your game, pass the user's email address and password to *Firebase Sign In With Email And Password*.



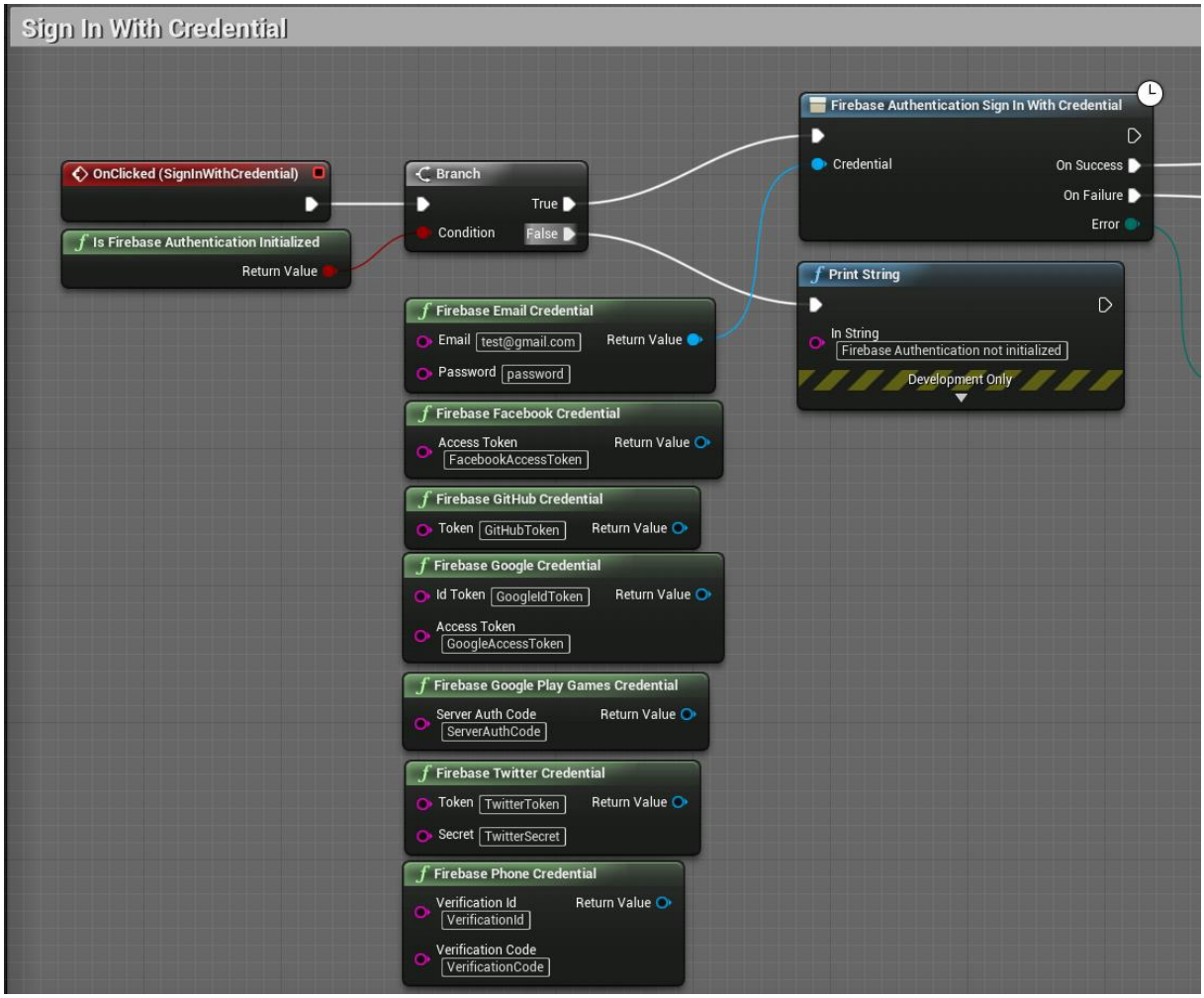
ii. Sign In Anonymously

You can use Firebase Authentication to create and use temporary anonymous accounts to authenticate with Firebase. These temporary anonymous accounts can be used to allow users who haven't yet signed up to your game to work with data protected by security rules. If an anonymous user decides to sign up to your game, you can link their sign-in credentials to the anonymous account so that they can continue to work with their protected data in future sessions.



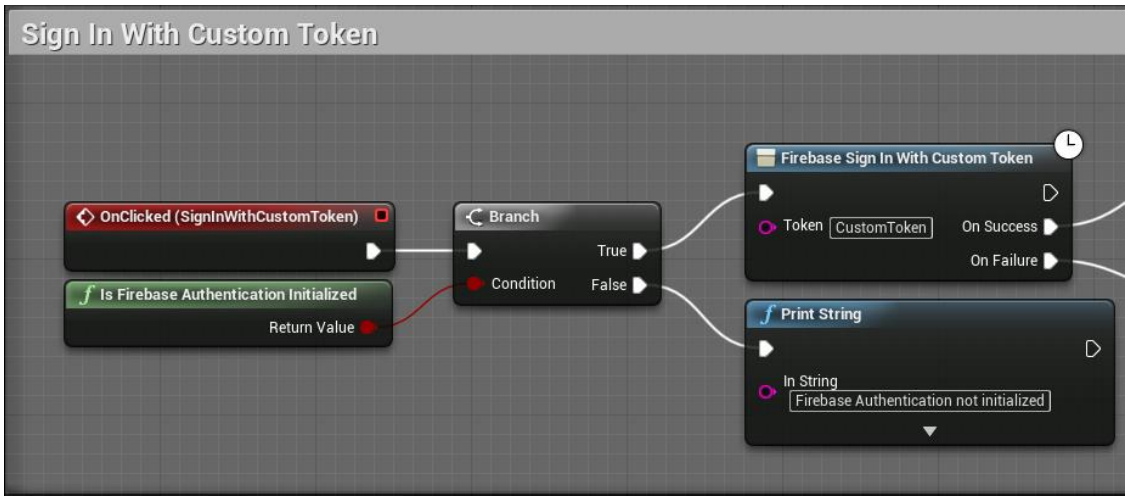
iii. Sign In With Credential

After a user signs in for the first time, a new user account is created and linked to the credentials – that is, the user name and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every game in your project, regardless of how the user signs in.



iv. Sign In With Custom Token

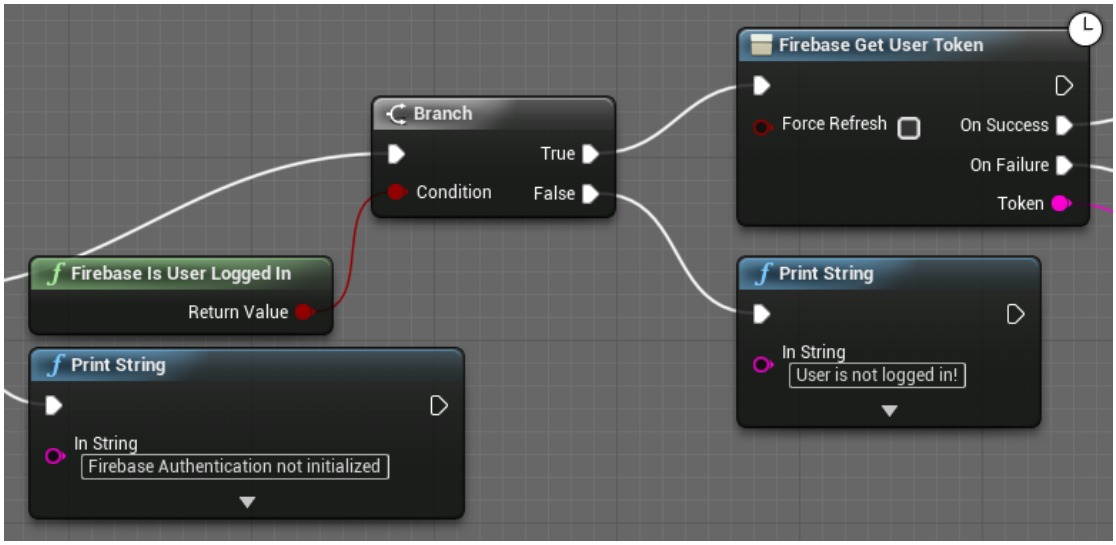
You can use your custom Auth Token authentication. An error is returned, if the token is invalid, expired or otherwise not accepted by the server.



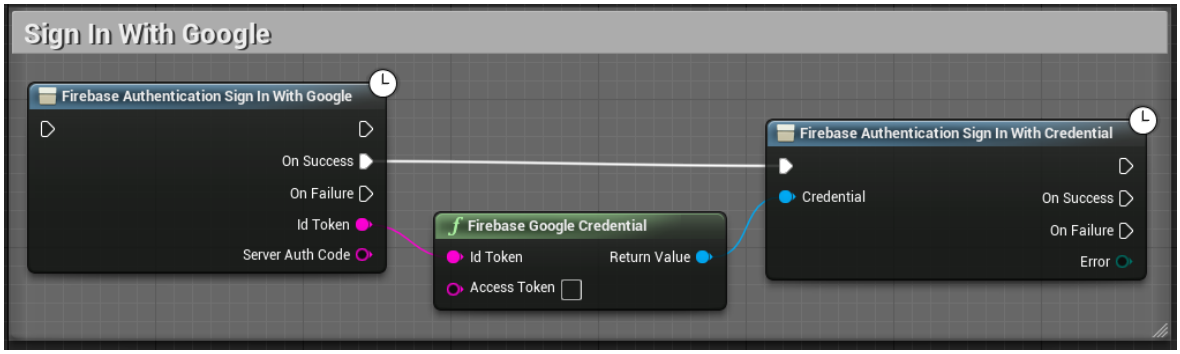
v. Get User Token

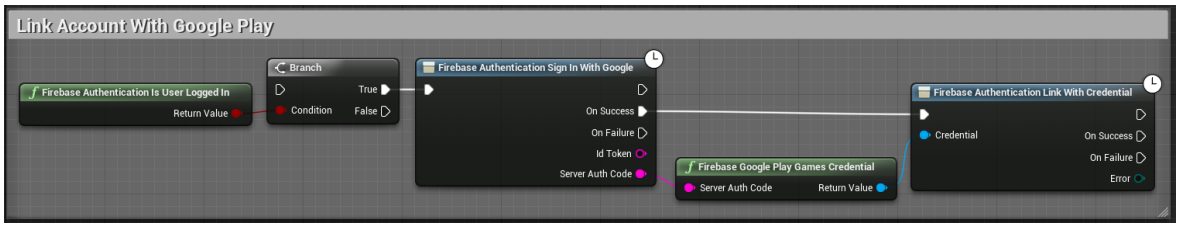
The Java Web Token (JWT) that can be used to identify the user to the backend.

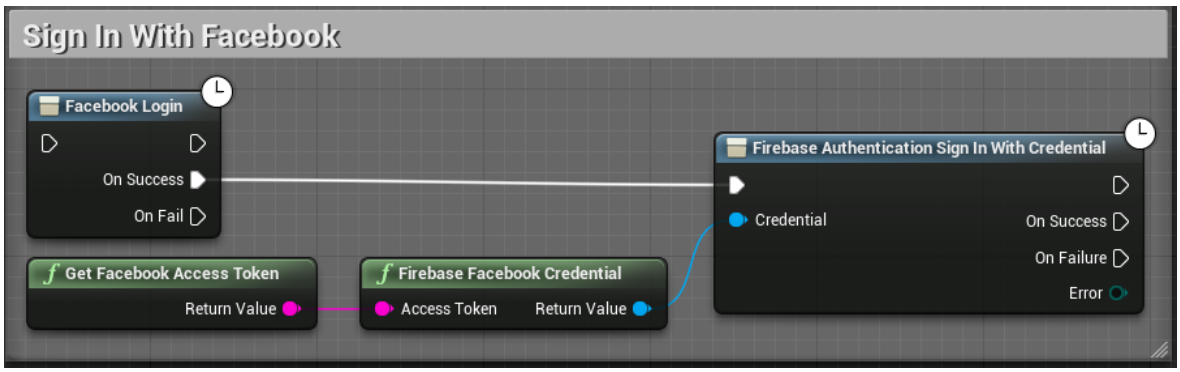
If a current ID token is still believed to be valid (i.e. it has not yet expired), that token will be returned immediately. You may set the optional *force_refresh* flag to get a new ID token, whether or not the existing token has expired. For example, you may use this when they have discovered that the token is invalid for some other reason.



vi. Examples of Sign In/Link With Credential



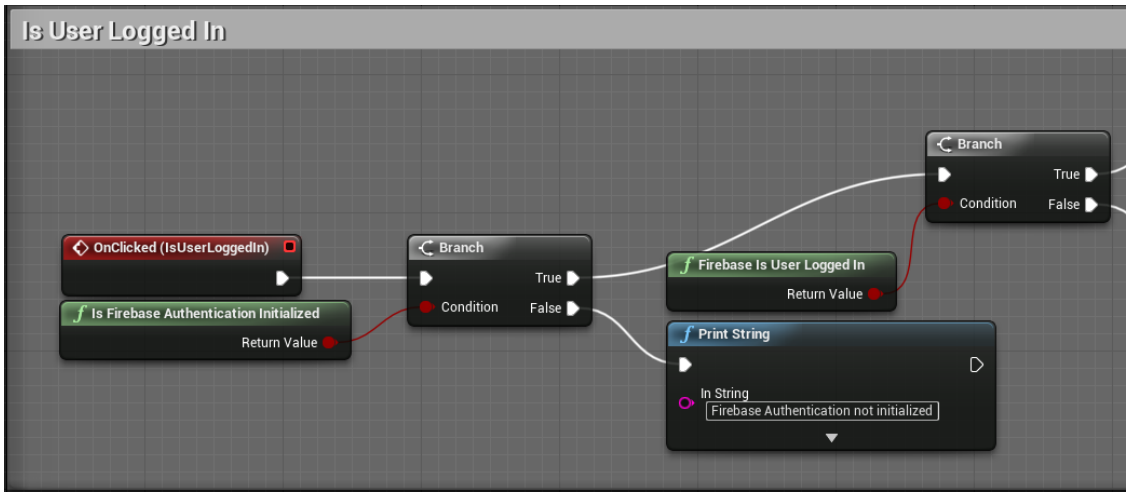




g. Managing Users

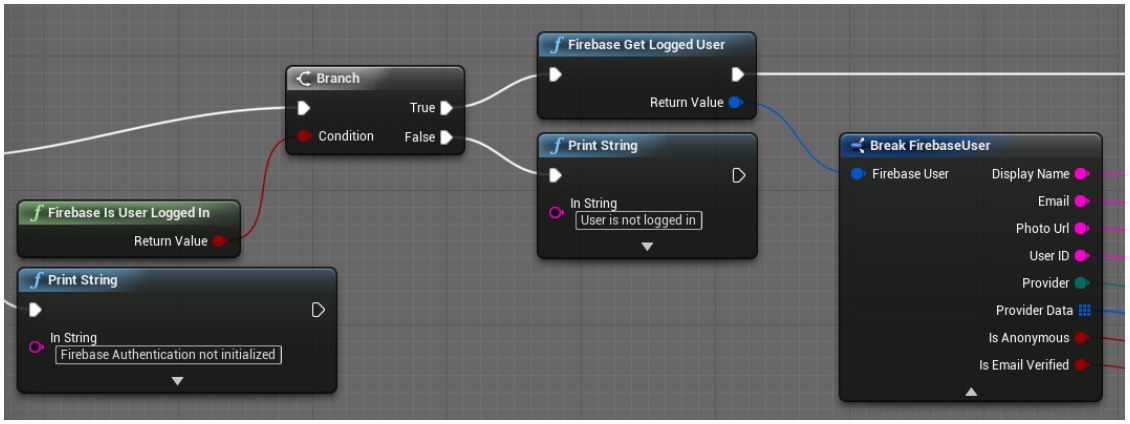
i. Is User Logged In

You can check whether a user is logged in to your game calling *Firebase Is User Logged In*.



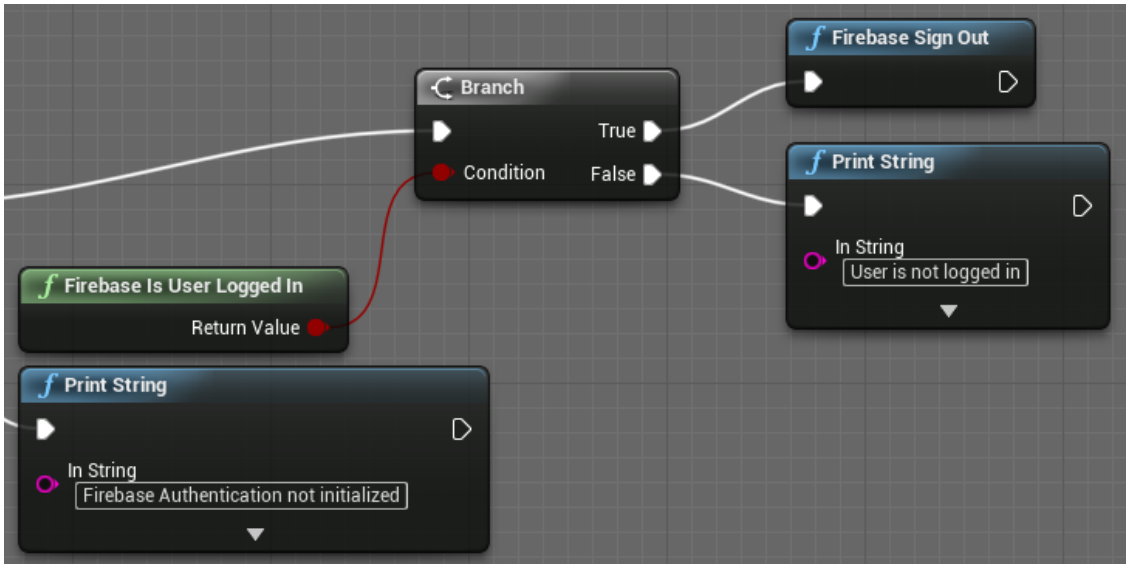
ii. Get Logged User

To get currently logged user's profile information, call *Firebase Get Logged User*.



iii. Sign Out

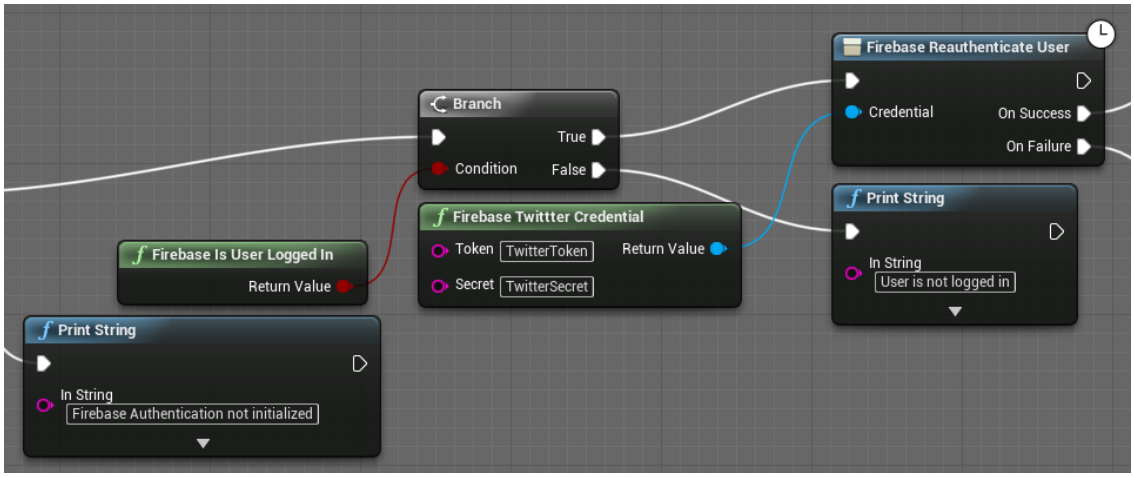
To sign out a user from the game, call the *Firebase Sign Out* function.



iv. Reauthenticate User

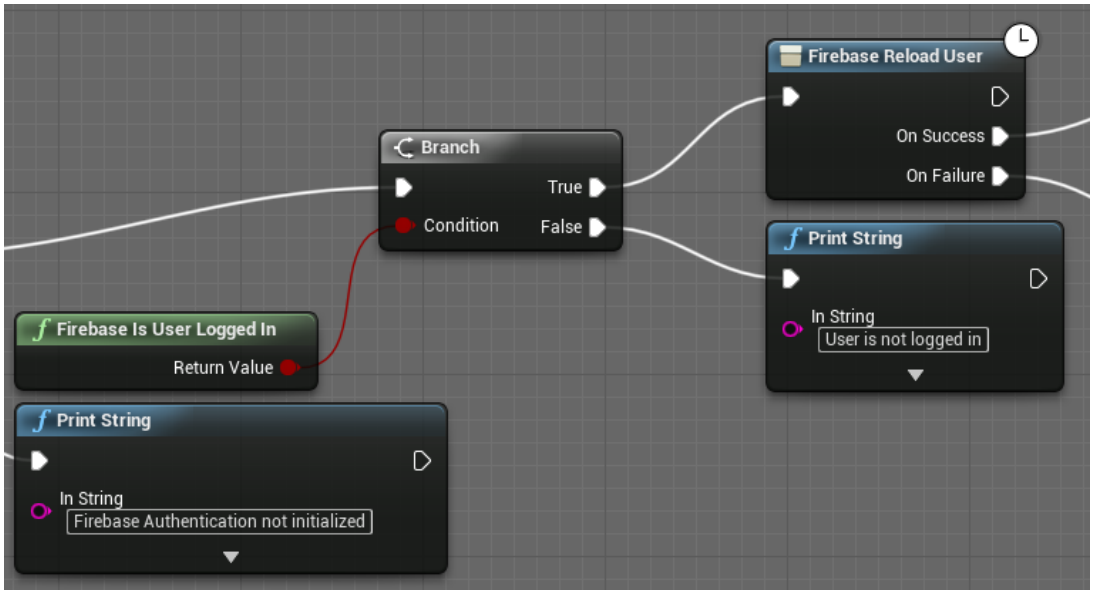
Some security-sensitive actions—such as deleting an account, setting a primary email address, and changing a password—require that the user has recently signed in. If you perform one of these actions, and the user signed in too long ago, the action fails.

When this happens, re-authenticate the user by getting new sign-in credentials from the user and passing the credentials to *Firebase Reauthenticate User*.



v. Reload User

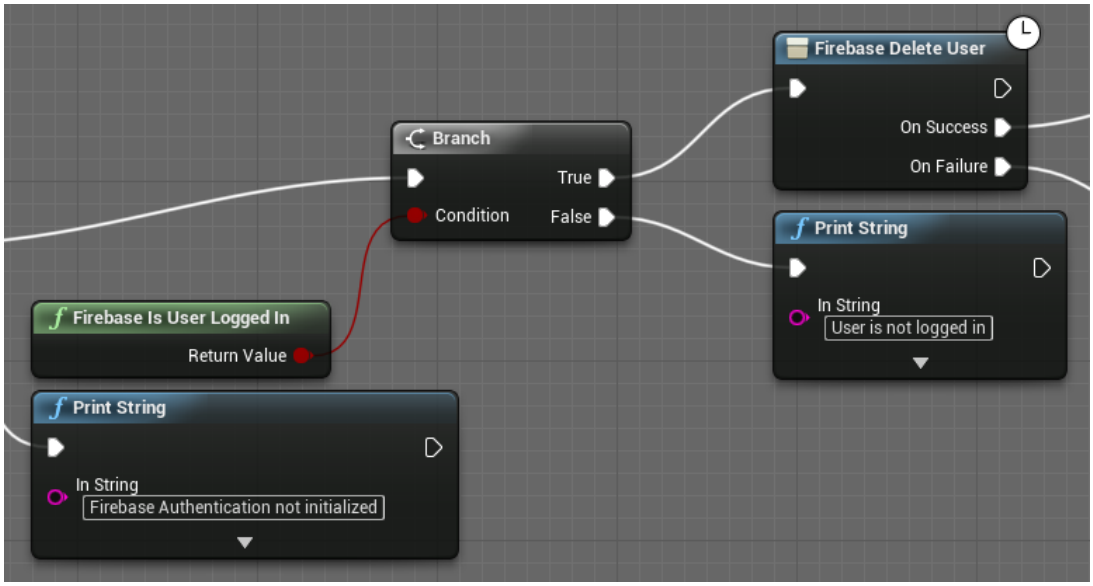
Firebase Reload User refreshes all the data for currently logged in user.



vi. Delete User

You can delete a user account with the *Firebase Delete User* method. You can also delete users from the Authentication section of the [Firebase console](), on the Users page.

Important: To delete a user, the user must have signed in recently.

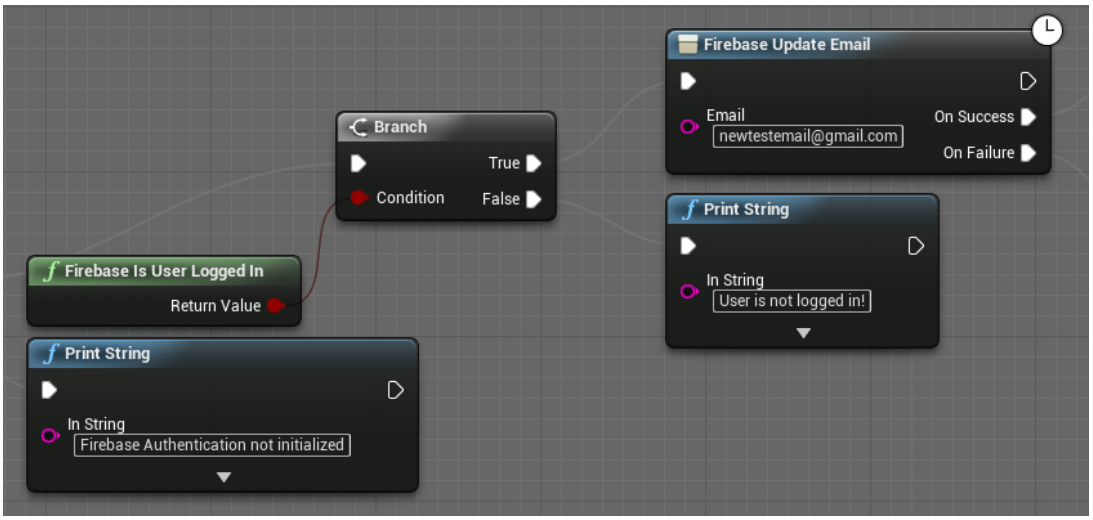


h. Update User Profile

i. Update Email

You can set a user's email address with the *Firebase Update Email* function.

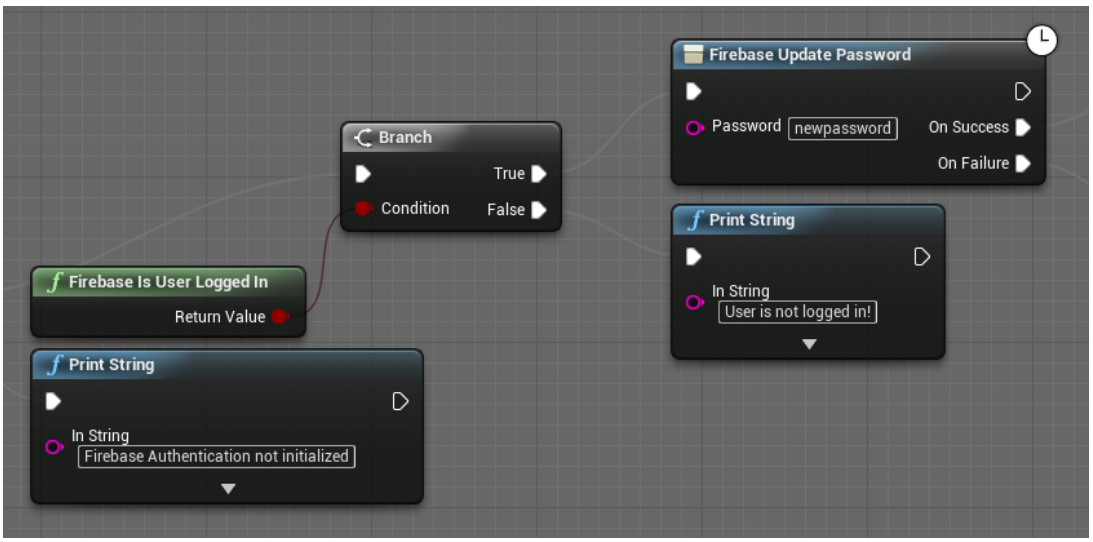
Important: To delete a user, the user must have signed in recently.



ii. Update Password

You can set a user's password with the *Firebase Update Password* function.

Important: To delete a user, the user must have signed in recently.

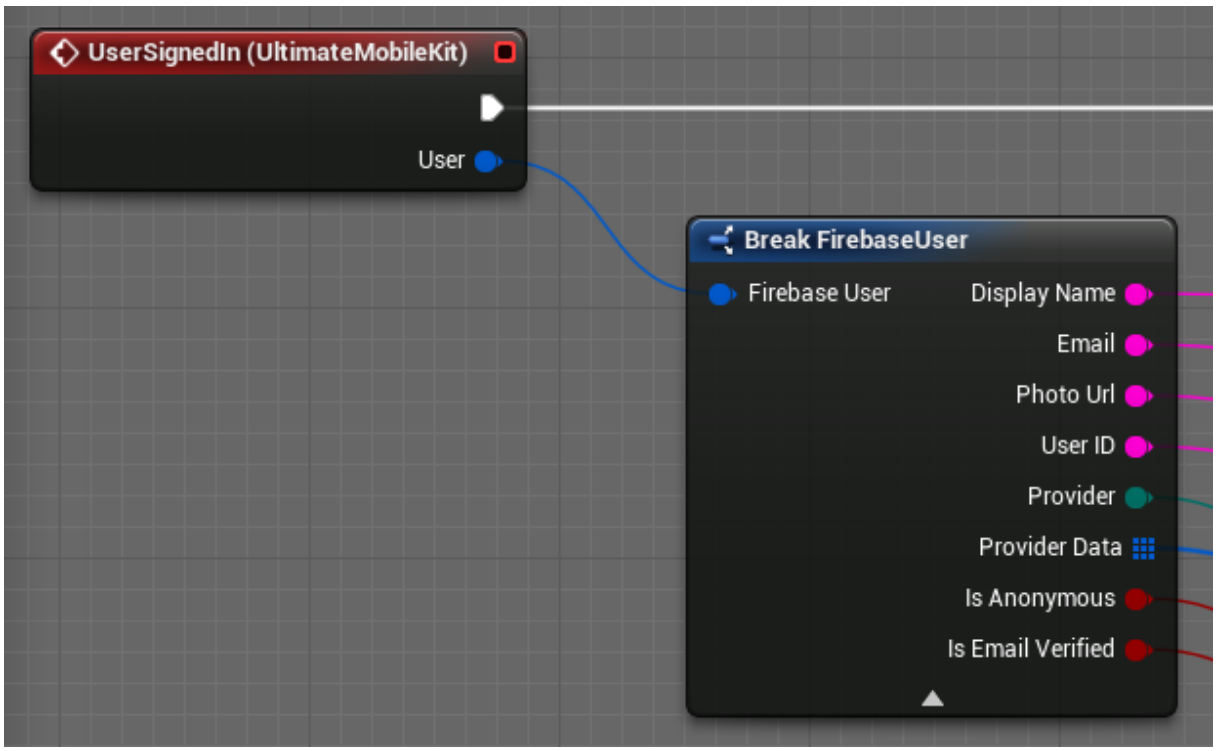


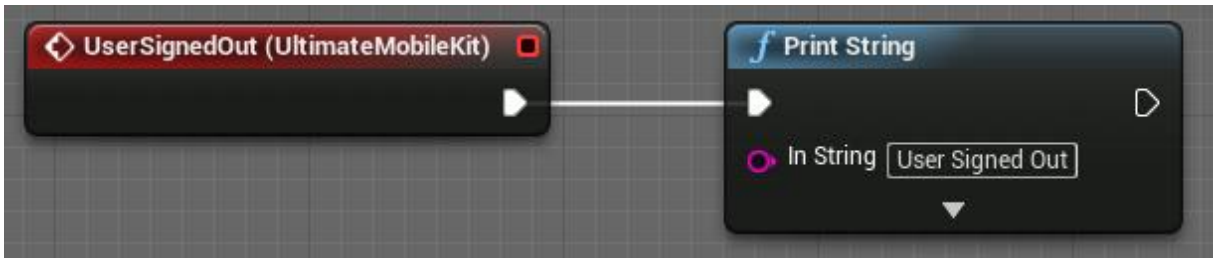
iii. Update User Profile

You can update a user's basic profile information – the user's display name and profile photo URL—with the *Firebase Update User Profile* function.

i. Authentication Listener

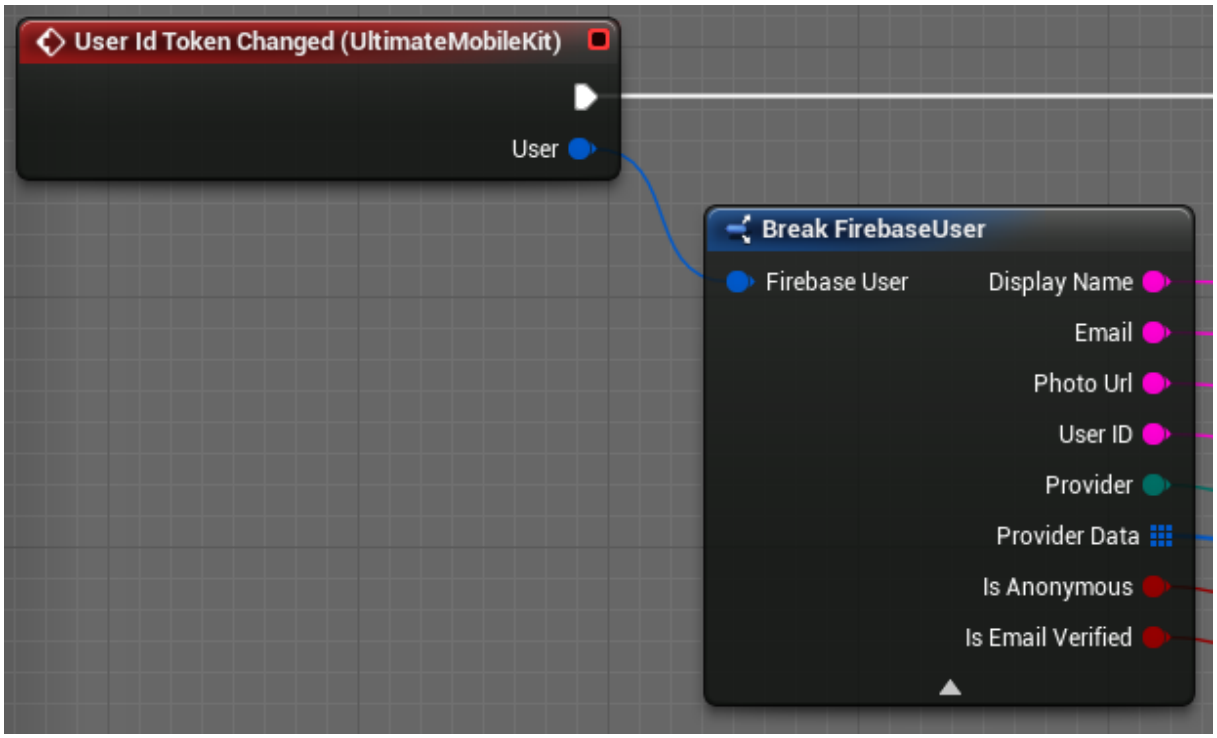
To respond to sign-in and sign-out events, attach the *UltimateMobileKit* component to the global actor (for example Game Mode). This listener gets called whenever the user's sign-in state changes. Because the listener runs only after the authentication object is fully initialized and after any network calls have completed, it is the best place to get information about the signed-in user. By using a listener, you ensure that the Auth object isn't in an intermediate state — such as initialization — when you get the current user.





j. User ID Token Changed Listener

Sometimes user's ID token might be changed. Event *User Id Token Changed* is called when there is a change in the current user's ID token.



11. Instance Id

Firebase Instance ID provides a unique ID per instance of your apps. In addition to providing unique IDs for authentication, Instance ID can generate security tokens for use with other services.

Key capabilities:

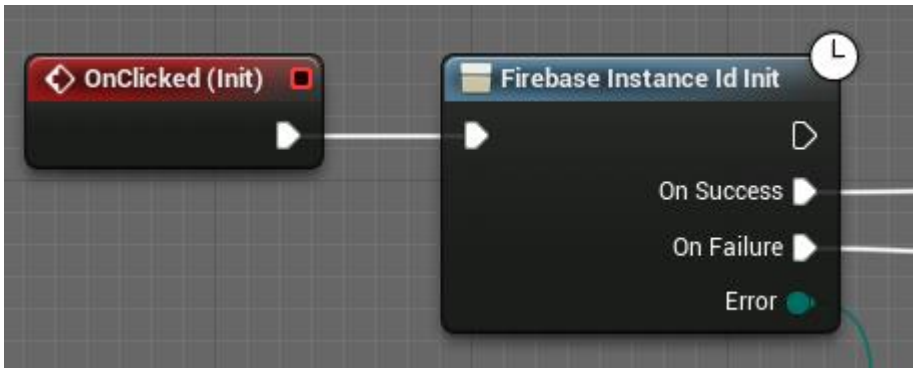
- **Identify and track games** - Instance ID is unique across all game instances across the world, so your database can use it to uniquely identify and track game instances. Your server-side code can verify, via the Instance ID cloud service, that an Instance ID is genuine and is the same ID as the original game that registered with your server. For privacy, your game can delete an Instance ID so it is no longer associated with any history in the database. The next time your game calls Instance ID it will get an entirely new Instance ID with no relationship to its previous one.
- **Generate Security Tokens** - Instance ID provides a simple API to generate security tokens that authorize third parties to access your game's server side managed resources.
- **Verify app authenticity** - pass Instance ID tokens to your server and use the Instance ID service to verify the game package name and check if it has a valid signature. Verifying tokens with the Instance ID Cloud Service helps identify known games. To reduce cost and redundant round trip communications, configure your server to store these tokens so the check is needed only once. In the event of a security concern, your game can delete tokens or Instance ID itself, and generate new ones. In addition, the Instance ID server initiates token or Instance ID refresh if it detects bugs or security issues. Verifying game authenticity is available only for games distributed by Google Play.
- **Confirm game device is active -** the Instance ID server can tell you when the device on which your game is installed was last used. Use this to decide whether to keep data from your game or send a push message to reengage with your users.

The Instance ID is long lived, but may expire for the following reasons:

- Device factory reset.
- User uninstalls the game.
- User performs *Clear Data* in the game.
- Device unused for an extended period (device and region determines the timespan).
- Instance ID service detects abuse or errors and resets the Instance ID.
- Server-side code if your client app requires that functionality.

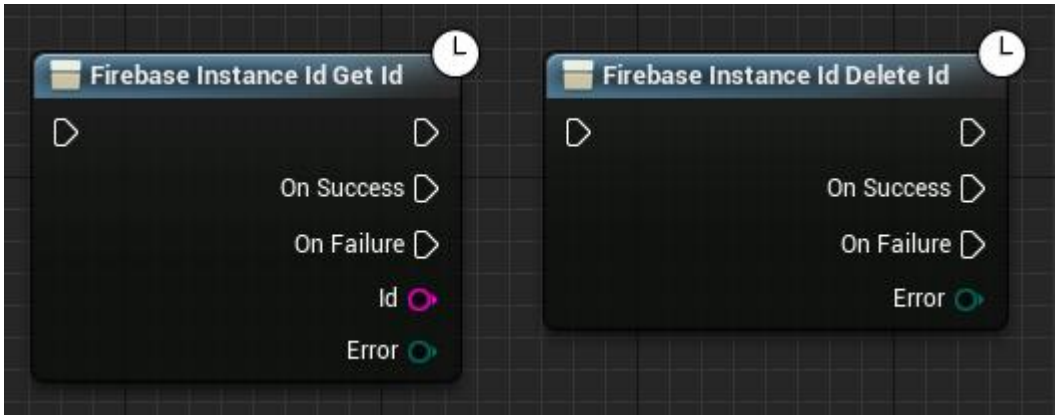
a. Initialization

Before making any call to the Firebase Instance Id you should first execute *Firebase Instance Id Init* function. If the result is a fail, study logs to find out what causes an issue.



b. Instance Id

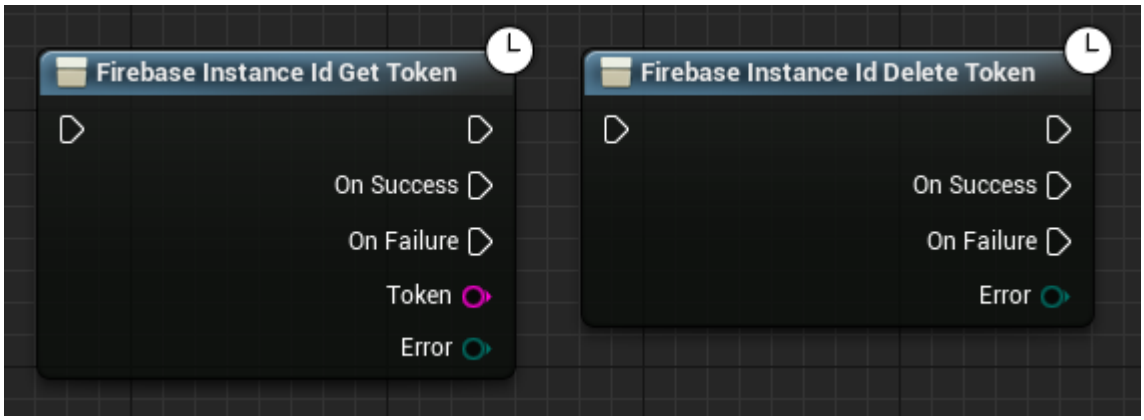
Firebase Instance Id Get Id returns a stable identifier that uniquely identifies the game instance. Once an Instance ID is generated, the library periodically sends information about the application and the device where it's running to the Firebase backend.



Firebase Instance Id Delete Id stops the periodic sending of data to the Firebase backend started when the Instance ID was generated, unless another library that requires Instance Id (like Firebase Cloud Messaging, Firebase Remote Config or Firebase Analytics) is used or it's configured to be executed automatically.

c. Tokens

To prove ownership of Instance ID and to allow servers to access data or services associated with the game call *Firebase Instance Id Get Token*. If you no longer need generated token call *Firebase Instance Id Delete Token* which revokes access to a scope for an entity.



12. Cloud Storage

Cloud Storage is built for game developers who need to store and serve user-generated content, such as save games, photos or videos.

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase games, regardless of network quality. You can use Google's SDKs to store saved games, images, audio, video, or other user-generated content. On the server, you can use Google Cloud Storage, to access the same files.

Developers use the Firebase SDKs for Cloud Storage to upload and download files directly from clients. If the network connection is poor, the client is able to retry the operation right where it left off, saving your users time and bandwidth.

Cloud Storage stores your files in a Google Cloud Storage bucket, making them accessible through both Firebase and Google Cloud. This allows you the flexibility to upload and download files from mobile clients via the Firebase SDKs, and do server-side processing such as image filtering or video transcoding using Google Cloud Platform. Cloud Storage scales automatically, meaning that there's no need to migrate to any other provider.

The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and Google provides a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want.

Key capabilities:

- **Robust operations** - Firebase SDKs for Cloud Storage perform uploads and downloads regardless of network quality. Uploads and downloads are robust, meaning they restart where they stopped, saving your users time and bandwidth.
- **Strong security** - Firebase SDKs for Cloud Storage integrate with Firebase Authentication to provide simple and intuitive authentication for developers. You can use our declarative security model to allow access based on filename, size, content type, and other metadata.
- **High scalability** - Cloud Storage for Firebase is built for exabyte scale when your app goes viral. Effortlessly grow from prototype to production using the same infrastructure that powers Spotify and Google Photos.

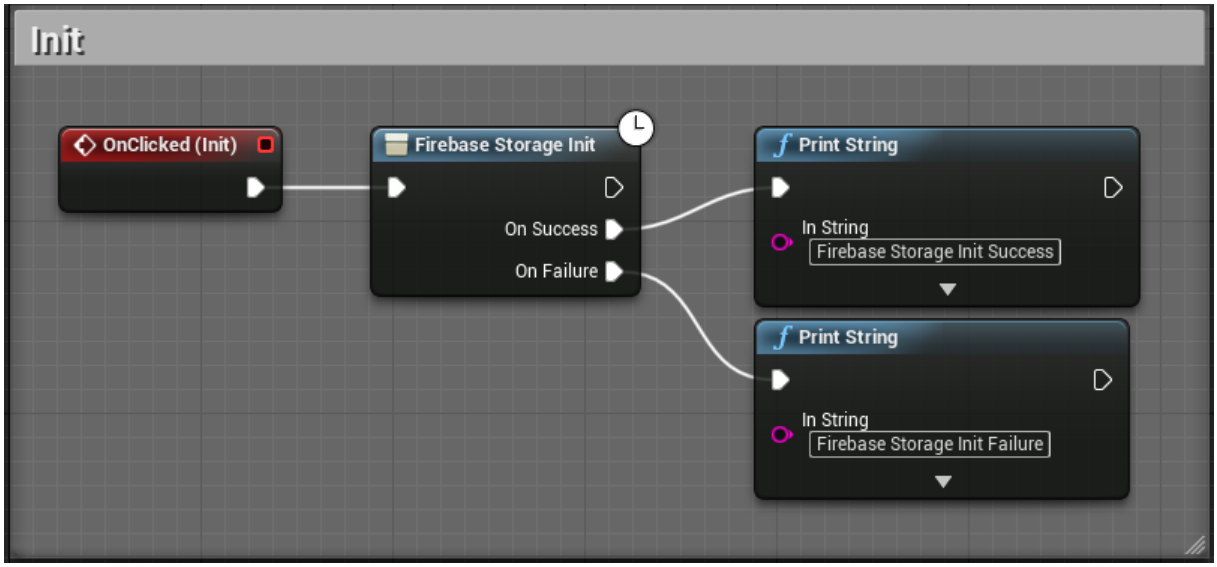
Official Firebase Cloud Storage documentation:
https://firebase.google.com/docs/storage/

Security & Rules: https://firebase.google.com/docs/storage/security/

Video introduction to Firebase Cloud Storage: https://youtu.be/_tyjqozrEPY

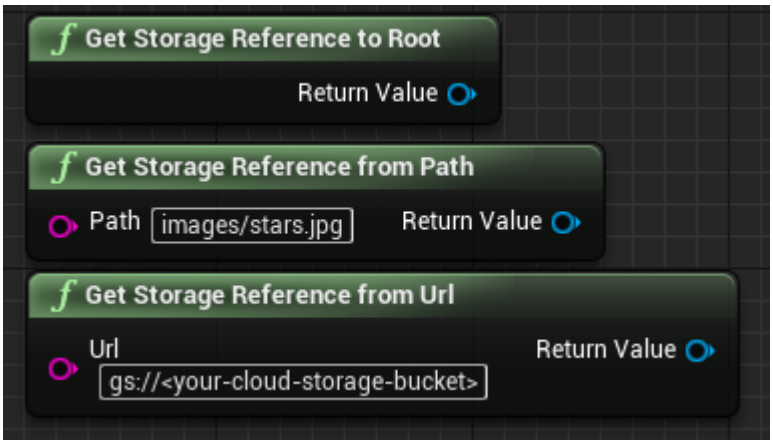
a. Initialization

Before making any call to the Firebase Cloud Storage you should first execute *Firebase Storage Init* function. If the result is a fail, study logs to find out what causes an issue.



b. Storage Reference

Your files are stored in a Google Cloud Storage bucket. The files in this bucket are presented in a hierarchical structure, just like the file system on your local hard disk. By creating a reference to a file, your game gains access to it. These references can then be used to upload or download data, get or update metadata or delete the file. A reference can either point to a specific file or to a higher level node in the hierarchy.

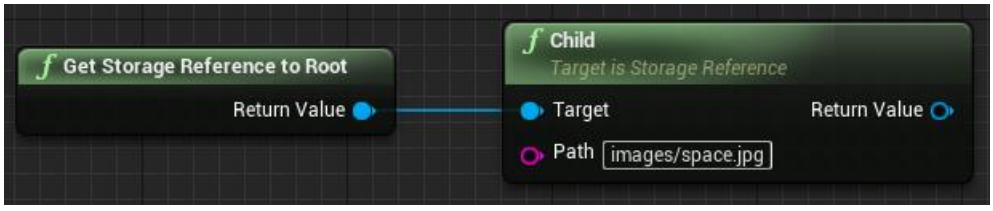


Create a reference to upload, download, or delete a file, or to get or update its metadata. A reference can be thought of as a pointer to a file in the cloud. References are lightweight, so you can create as many as you need. They are also reusable for multiple operations.

References are created from the storage service on your Firebase game by calling one of three functions:

- *Get Storage Reference to Root* – reference to the highest level of your storage
- *Get Storage Reference from Path* – reference to the specified path from root
- *Get Storage Reference from Url* – URL of the form *gs://<your-cloud-storage-bucket>*. You can find this URL in the Storage section of the Firebase console.

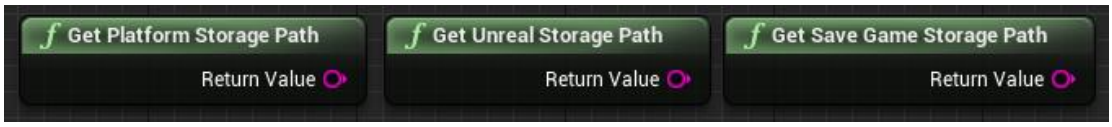
You can create a reference to a location lower in the tree, say *images/space.jpg*, by using the child method on an existing reference.



You can also use the *Parent* function to navigate up one level in your file hierarchy.

Child and *Parent* can be chained together multiple times, as each returns a reference. The exception is the *Parent* of *Root*, which is an invalid *Storage Reference*.

c. Storage Path



For downloading or uploading files you need to provide location where file should be located on the iOS or Android device. You can do this using the following functions:

- *Get Platform Storage Path* – returns a path to the physical storage on a device
- *Get Unreal Storage Path* – returns a path to the main directory of Unreal filesystem on a device
- *Get Save Game Storage Path* - returns a path to the directory of Unreal Save Games on a device

d. Download File

Cloud Storage allows developers to quickly and easily download files from a Google Cloud Storage bucket provided and managed by Firebase.

Note: By default, Cloud Storage buckets require Firebase Authentication to download files. You can change your Firebase Security Rules for Cloud Storage to allow unauthenticated access.

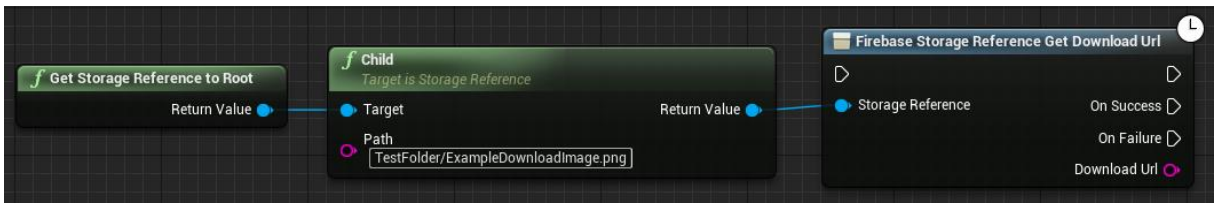
To download a file, first create a Cloud Storage reference to the file you want to download.

You can create a reference by appending child paths to the storage root, or you can create a reference from an existing gs:// or https:// URL referencing an object in Cloud Storage.

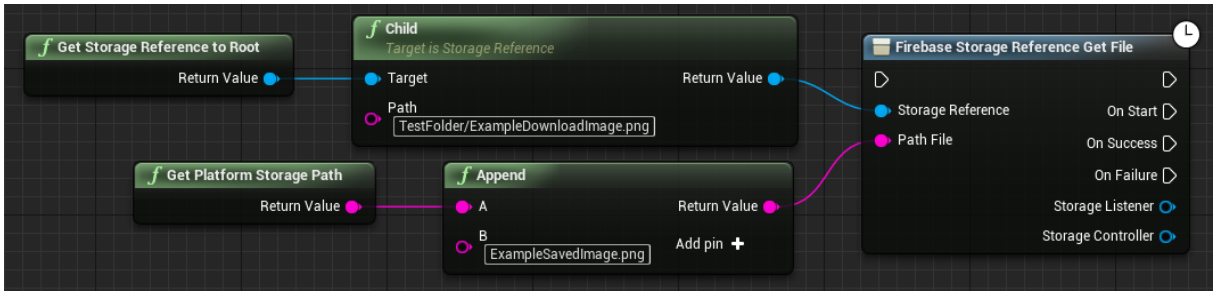
Once you have a reference, you can download files from Cloud Storage in two ways:

- Generate a string URL representing the file online
- Download to a specific path on the device

If you already have download infrastructure based around URLs, or just want a URL to share, you can get the download URL for a file by calling the *Firebase Storage Reference Get Download Url* function on a storage reference.



The *Firebase Storage Reference Get File* function downloads a file directly to a local device. Use this, if your users want to have access to the file while offline or to share in a different app.



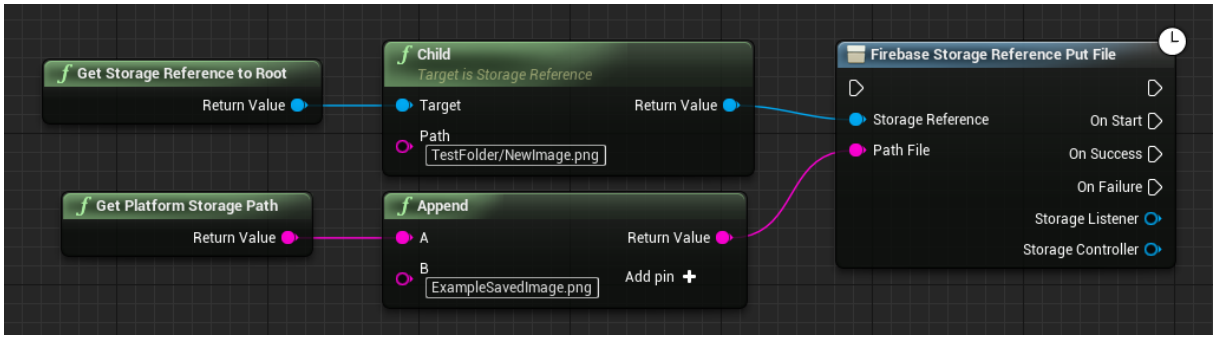
e. Upload File

Cloud Storage allows developers to quickly and easily upload files to a Google Cloud Storage bucket provided and managed by Firebase.

Note: By default, Cloud Storage buckets require Firebase Authentication to upload files. You can change your Firebase Security Rules for Cloud Storage to allow unauthenticated access.

To upload a file, first create a Cloud Storage reference to the location in Cloud Storage you want to upload the file to. You cannot upload data with a reference to the root of your Google Cloud Storage bucket. Your reference must point to a child URL. Once you have a reference, you can upload files to Cloud Storage.

You can upload local files on the devices, such as photos and videos from the camera or saved games, with the *Firebase Storage Reference Put File* function. You can also include metadata when you upload files (more in the Get/Update Metadata section).

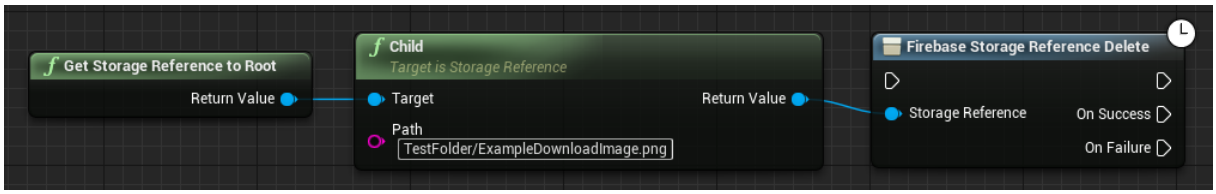


f. Delete File

Of course, you can also delete files from Cloud Storage.

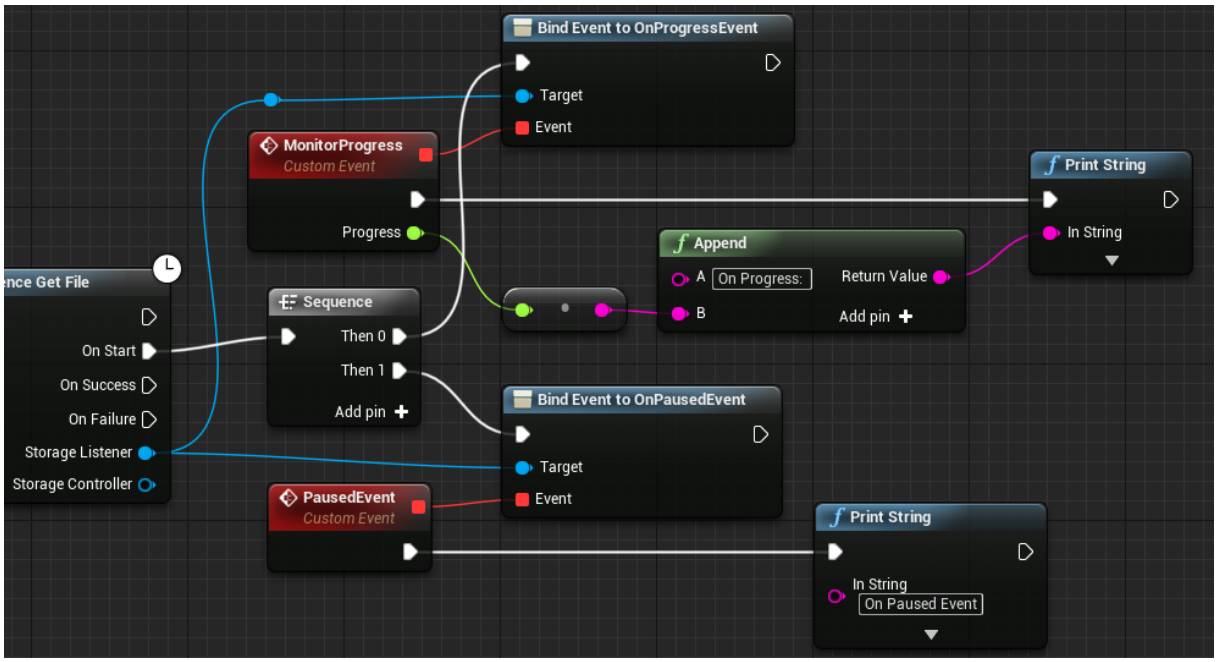
Note: By default, Cloud Storage buckets require Firebase Authentication to delete files. You can change your Firebase Security Rules for Cloud Storage to allow unauthenticated access.

To delete a file, first create a reference to that file. Then call the *Firebase Storage Reference Delete* method on that reference.



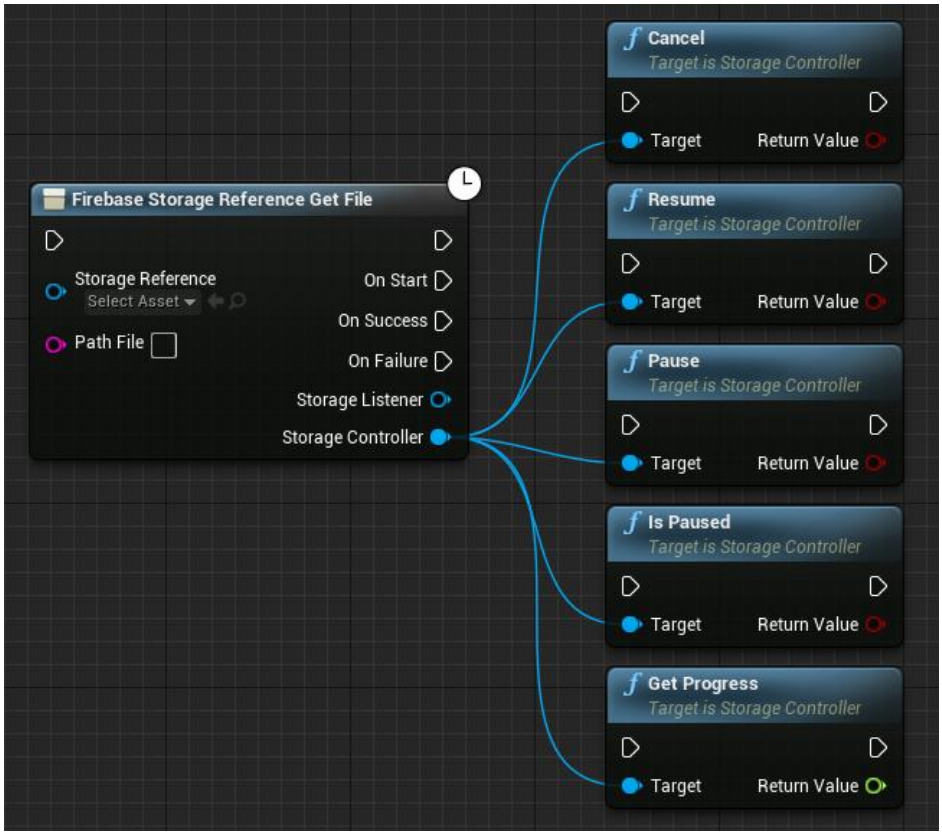
g. Monitor progress of Downloads/Uploads

Downloads/Uploads functions return the *Storage Listener* object, to which you can bind *OnProgressEvent* and *OnPausedEvent* in order to monitor the progress of files tasks.



h. Manage Downloads/Uploads

In addition to starting downloads/uploads, you can pause, resume, and cancel downloads/uploads using the *Pause*, *Resume*, and *Cancel* functions on *Storage Controller*.

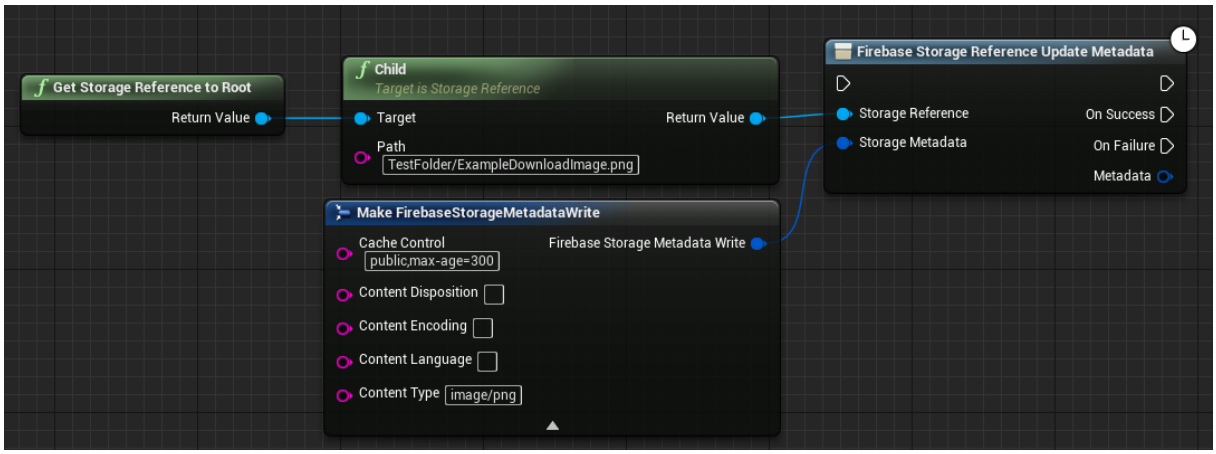


i. Get/Update Metadata

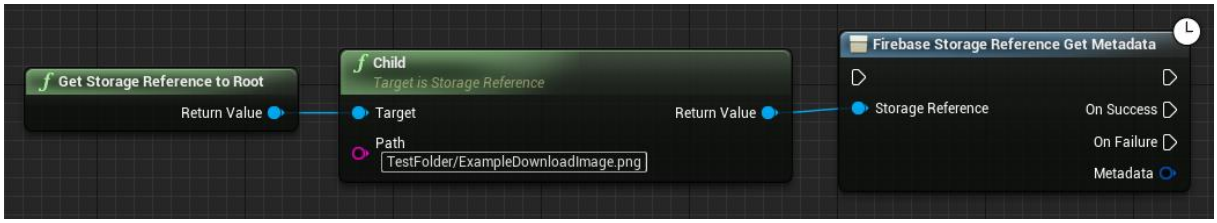
After uploading a file to Cloud Storage reference, you can also get and update the file metadata, for example to update the content type.

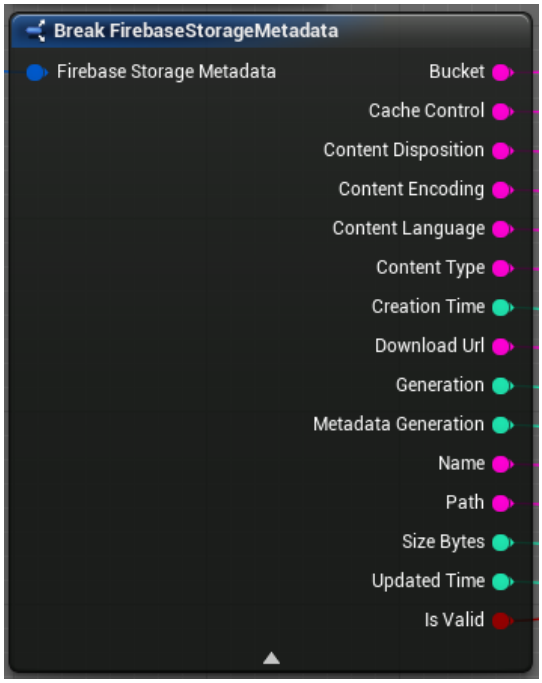
Note: By default, Cloud Storage buckets require Firebase Authentication to delete files. You can change your Firebase Security Rules for Cloud Storage to allow unauthenticated access.

You can update file metadata at any time after the file upload completes by using the *Firebase Storage Reference Update Metadata* function. Only the properties specified in the metadata are updated, all others are left unmodified. You can delete writable metadata properties by passing the empty string.



File metadata contains common properties such as name, size, and content_type (often referred to as MIME type) in addition to some less common ones like content_disposition and time_created. This metadata can be retrieved from a Cloud Storage reference using the *Firebase Storage Reference Get Metadata* function.





13. Remote Config

Firebase Remote Config is a cloud service that lets you change the behavior and appearance of your game without requiring users to download a game update. When using Remote Config, you create in-game default values that control the behavior and appearance of your game. Then, you can later use the Firebase console to override in-game default values for all game users or for segments of your userbase. Your game controls when updates are applied, and it can frequently check for updates and apply them with a negligible impact on performance.

Remote Config includes a client library that handles important tasks like fetching parameter values and caching them, while still giving you control over when new values are activated so that they affect your game's user experience. This lets you safeguard your game experience by controlling the timing of any changes.

The Remote Config client library get methods provide a single access point for parameter values. Your game gets service-side values using the same logic it uses to get in-game default values, so you can add the capabilities of Remote Config to your game without writing a lot of code.

To override in-game default values, you use the Firebase console to create parameters with the same names as the parameters used in your game. For each parameter, you can set a service-side default value to override the in-game default value, and you can also create conditional values to override the in-game default value for game instances that meet certain conditions.

Key capabilities:

- **Quickly roll out changes to your game's userbase** - you can make changes to your game's default behavior and appearance by changing service-side parameter values. For example, you could change your game's layout or color theme to support a seasonal promotion, with no need to publish an game update.
- **Customize your game for segments of your userbase** - you can use Remote Config to provide variations on your game's user experience to different segments of your userbase by game version, by Firebase Analytics audience, by language, and more.
- **Run A/B tests to improve your game** - you can use Remote Config random percentile targeting with Firebase Analytics to A/B test improvements to your game across different segments of your userbase so that you can validate improvements before rolling them out to your entire userbase.

Official Firebase Remote Config documentation:
https://firebase.google.com/docs/remote-config/

Parameters and Conditions: https://firebase.google.com/docs/remote-config/parameters

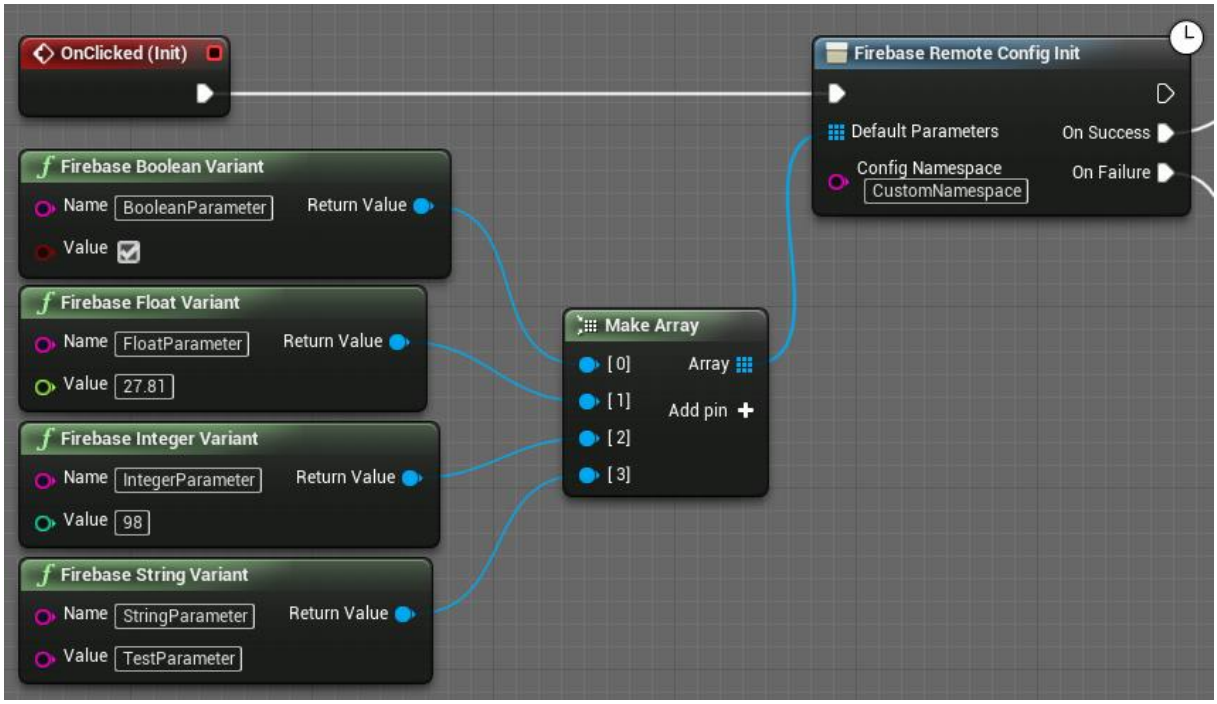
Video introduction to Firebase Remote Config: https://youtu.be/_CXXVFPO6f0

Note: Some Remote Config functions allow you to specify a namespace. These functions are reserved for future use. You do not need to specify a namespace to use Remote Config.

a. Initialization

Before making any call to the Firebase Remote Config you should first execute *Firebase Remote Config Init* function. If the result is a fail, study logs to find out what causes an issue.

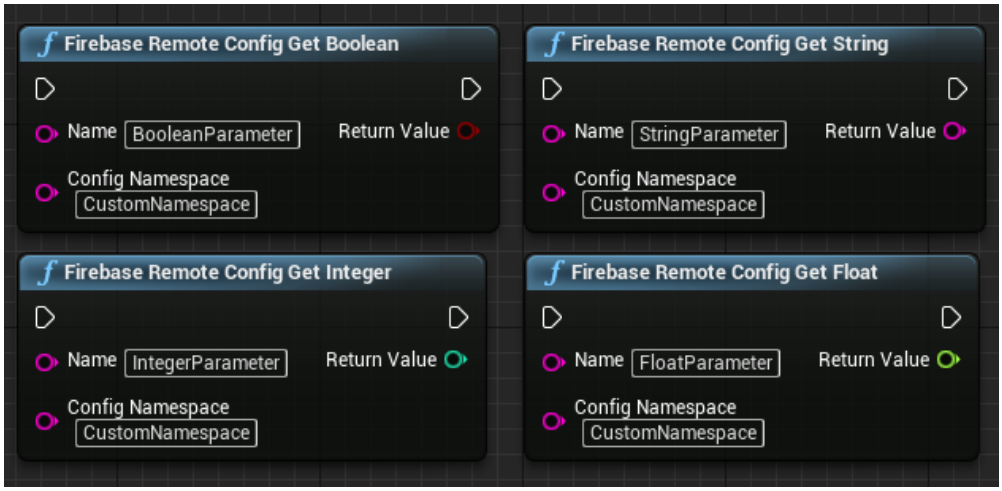
You can set in-app default parameter values in the Remote Config object, so that your game behaves as intended before it connects to the Remote Config service, and so that default values are available if none are set in the service. Make array with *Firebase Variant* parameters and pass to the *Firebase Remote Config Init* function.



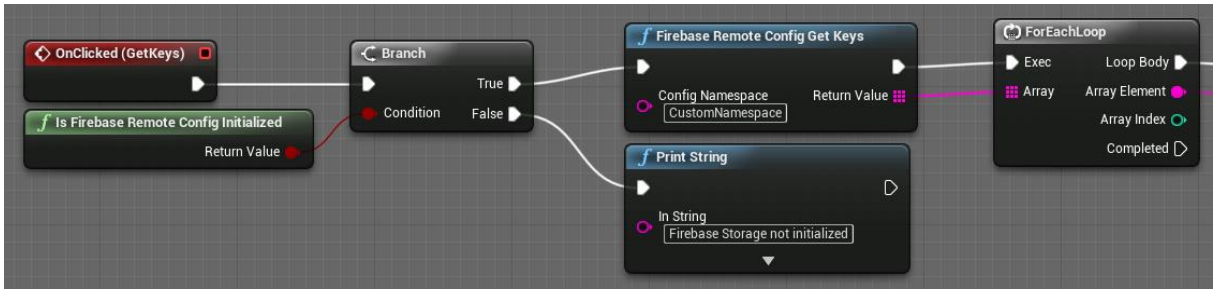
b. Get parameter values to use in your game

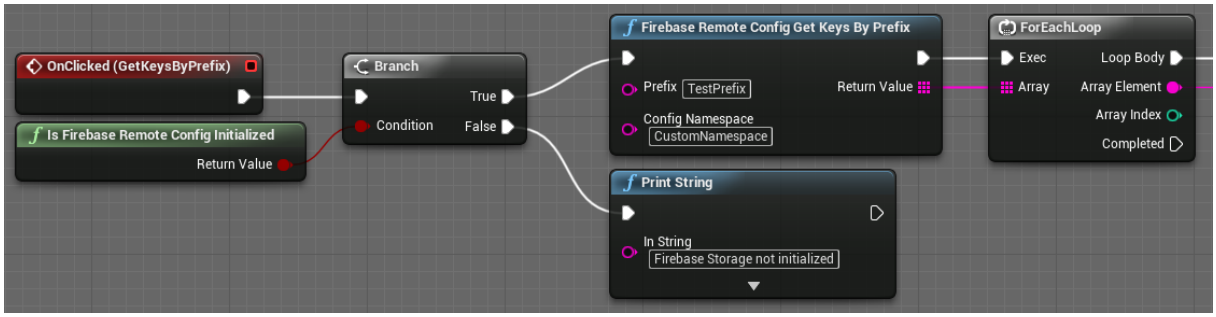
Now you can get parameter values from the Remote Config object. If you set values in the Remote Config service, fetched them, and then activated them, those values are available to your game. Otherwise, you get the in-app default parameter values configured using *Firebase Remote Config Init*.

To get these values, call the method listed below that maps to the data type expected by your game, providing the parameter key as an argument.



You can get the set of all keys (*Firebase Remote Config Get Keys*) or the set of that start with the given prefix (*Firebase Remote Config Get Keys By Prefix*).





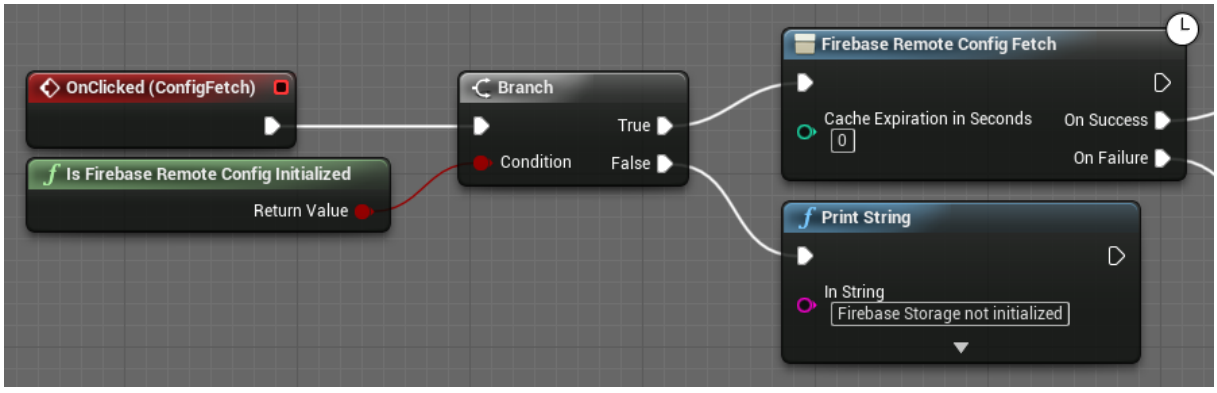
c. Set parameter values in the service (as needed)

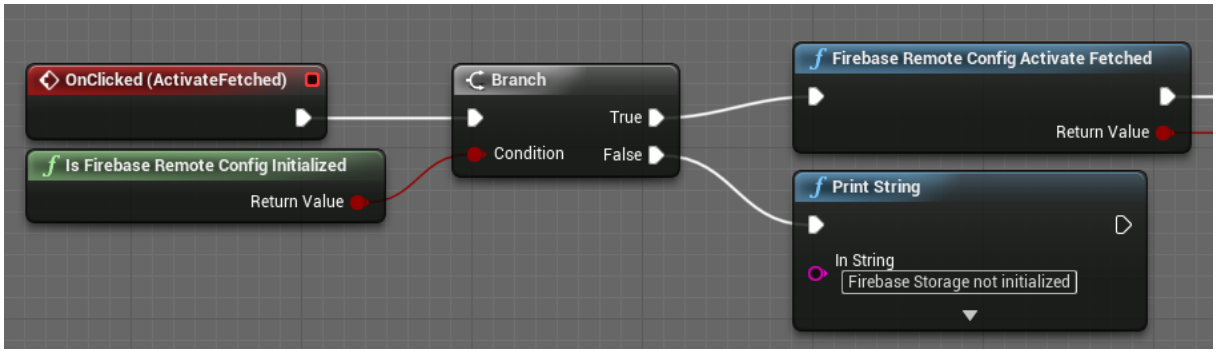
1. Open your project in the Firebase console.
2. Select *Remote Config* from the menu to view the Remote Config dashboard.
3. Define parameters with the same names as the parameters that you defined in your app. For each parameter, you can set a default value (which will eventually override the in-app default value) and conditional values. To learn more, see Remote Config parameters and conditions.

d. Fetch and activate values from the service (as needed)

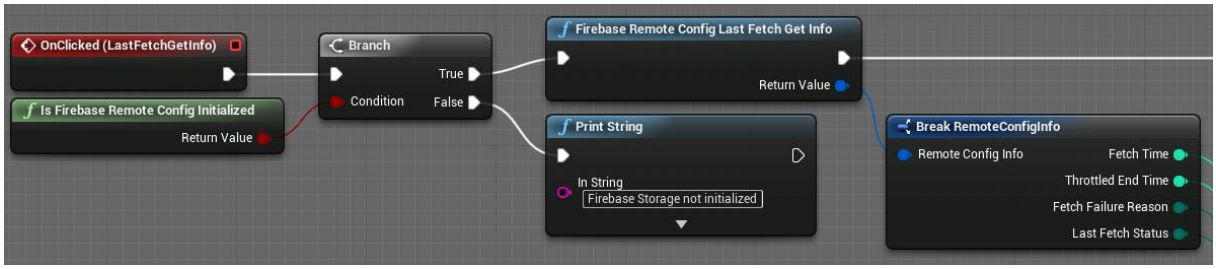
1. To fetch parameter values from the Remote Config service, call the *Firebase Remote Config Fetch* function. Any values that you set on the Remote Config Server are fetched and cached in the Remote Config object.
2. To make fetched parameter values available to your game, call the *Firebase Remote Config Activate Fetched* function.

Because these updated parameter values affect the behavior and appearance of your game, you should activate the fetched values at a time that ensures a smooth experience for your user, such as the next time that the user opens your game.





You can also get information about the last fetch calling *Firebase Remote Config Last Fetch Get Info*.



14. Performance Monitoring

Firebase Performance Monitoring is a service that helps you to gain insight into the performance characteristics of your iOS and Android games. You use the Performance Monitoring to collect performance data from your game, and then review and analyze that data in the Firebase console. Performance Monitoring helps you to understand where and when the performance of your game can be improved so that you can use that information to fix performance issues.

Key capabilities:

- **Automatically measure app startup time, HTTP/S network requests, and more** – when you integrate the Performance Monitoring into your iOS or Android game, you don't need to do anything before your game starts monitoring several critical aspects of game performance: startup time, activity while in the foreground, activity while in the background, and HTTP/S network requests.
- **Gain insight into situations where game performance could be improved** – optimizing the performance of your game can be challenging when you don't know exactly why it is falling short of user expectations. That's why Performance Monitoring lets you see performance metrics broken down by country, device, app version, and OS level.
- **Customize Performance Monitoring for your game** – you can create traces to capture your game's performance in specific situations, like when you load a new screen or level. And, you can create counters to count events that you define (like cache hits) during those traces.

Official Firebase Performance Monitoring documentation:
https://firebase.google.com/docs/perf-mon/

Video introduction to Firebase Performance Monitoring:
https://youtu.be/0EHSPFvH7vk

How does it work?

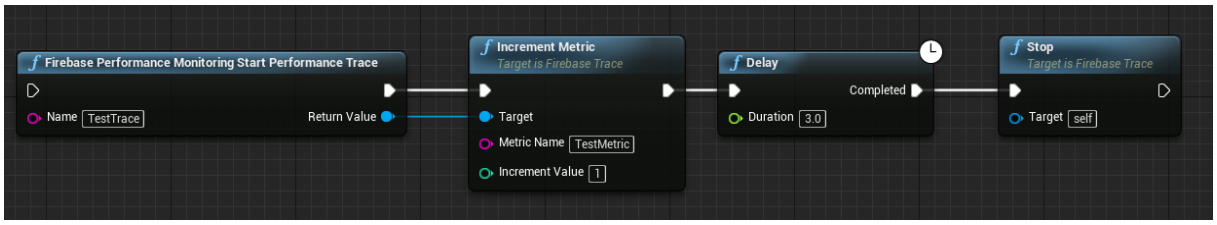
Performance Monitoring monitors traces and HTTP/S network requests in your game. A *trace* is a report of performance data captured between two points in time in your game. When installed, the Performance Monitoring automatically provides *app start* traces, which measure the time between when the user opens the game and when the game is responsive. It also provides *app in foreground* traces and *app in background*

traces to give you insight into how your game performs when in the foreground or when idle.

You can also configure *custom traces*. A custom trace is a report of performance data associated with some of the code in your game. You define the beginning and end of a custom trace using the functions provided by the Ultimate Mobile Kit. A custom trace can be further configured to record counters for performance-related events that occur within its scope. For example, you could create a counter for the number of cache hits and misses or the number of times that the UI becomes unresponsive for a noticeable period of time.

An *HTTP/S network request* is a report that captures the time between when your game issues a request to a service endpoint and when the response from that endpoint is complete. For any endpoint that your games makes a request to, the Ultimate Mobile Kit will capture several metrics:

- **Response time:** Time between when the request is made and when the response is fully received
- **Payload size:** Byte size of the network payload downloaded and uploaded by the game
- **Success rate:** Percentage of successful responses compared to total responses (to measure network or server failures)



1. Just before the point where you want to start a trace in your game, add the *Firebase Start Performance Trace* function to start a trace called *TestTrace*.
2. To count performance-related events that occur in your game (such as cache hits and misses), add the *Increment Metric* function each time that the event occurs.
3. Just after the point where you want to stop your trace, add the *Stop* function.

15. Crashlytics

Get clear, actionable insight into app issues with this powerful crash reporting solution for Android and iOS.

Firebase Crashlytics is a lightweight, realtime 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.

Key capabilities:

- **Curated crash reports** - Crashlytics synthesizes an avalanche of crashes into a manageable list of issues, provides contextual information, and highlights the severity and prevalence of crashes so you can pinpoint the root cause faster.
- **Cures for the common crash** - Crashlytics offers Crash Insights, helpful tips that highlight common stability problems and provide resources that make them easier to troubleshoot, triage, and resolve.
- **Integrated with Analytics** - Crashlytics can capture your game's errors as app_exception events in Analytics. The events simplify debugging by giving you access a list of other events leading up to each crash, and provide audience insights by letting you pull Analytics reports for users with crashes.
- **Realtime alerts** - get realtime alerts for new issues, regressed issues, and growing issues that might require immediate attention.

Official Firebase Crashlytics documentation:
https://firebase.google.com/docs/crashlytics/

Video introduction to Firebase Crashlytics: https://youtu.be/k_mdNRZzd30

a. Force a crash to test your implementation

You don't have to wait for a crash to know that Crashlytics is working. You can use the functions to force a crash by calling *Firebase Crashlytics Force Crash* or *Firebase Crashlytics Force Exception*.

When testing, reopen your game after crash to make sure Crashlytics has a chance to report the crash. The report should appear in the Firebase console within five minutes.

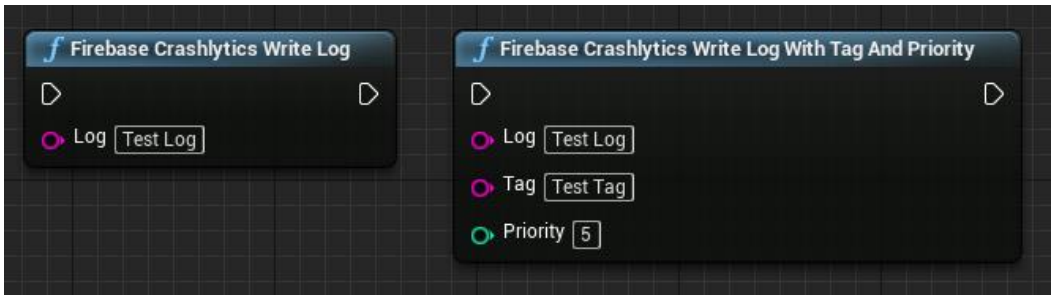
b. Customize crash reports

Firebase Crashlytics can work with very little setup on your part - as soon as you enable module, Crashlytics gets to work sending crash reports to the Firebase console.

For more fine-grained control of your crash reports, customize your Crashlytics configuration. For example, you can enable opt-in reporting for privacy-minded users, add logs to track down pesky bugs, and more.

i. Add custom logs

To give yourself more context for the events leading up to a crash, you can add custom Crashlytics logs to your app. Crashlytics associates the logs with your crash data and makes them visible in the Firebase console.



ii. Add custom keys

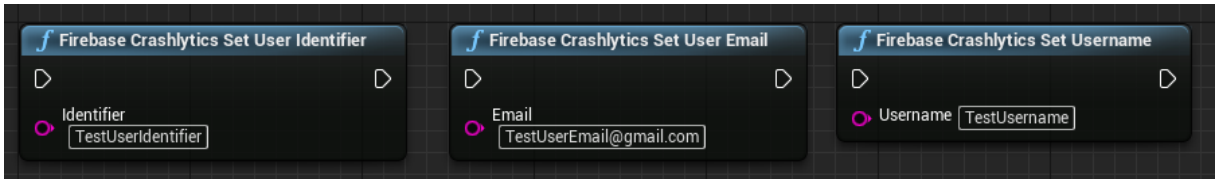
Custom keys help you get the specific state of your game leading up to a crash. You can associate arbitrary key/value pairs with your crash reports, and see them in the Firebase console.

There are four methods to set keys. Each handles a different data type.

iii. Set user data

To diagnose an issue, it's often helpful to know which of your users experienced a given crash. Crashlytics includes a way to anonymously identify users in your crash reports.

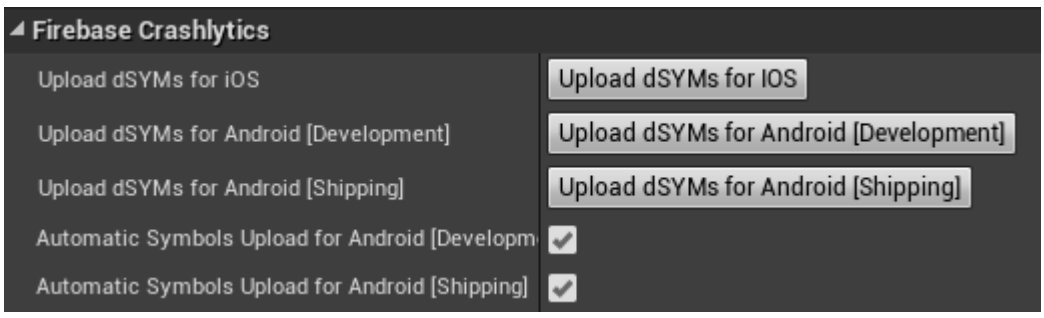
To add user IDs to your reports, assign each user a unique identifier in the form of an ID number, token, or hashed value.



If you need to clear a user identifier after you set it, reset the value to a blank string.

c. Upload dSYMs symbols

Firebase Crashlytics needs your debug symbols (dSYMs) to give you deobfuscated, human-readable crash reports. Our plugin offers automatic or manual uploading dSYMs to Crashlytics by Plugin Settings.



If you choose option *Automatic Symbols Upload for Android [Development]* or *Automatic Symbols Upload for Android [Shipping]*, all symbols will be automatically uploaded after every build (please note it increases building times!). Alternatively, you can manually upload symbols when needed. Please note you should upload dSYMs after each build which is intended to recognize by Firebase Crashlytics (development and shipping builds have different symbols!).

i. Android

1. Build your game in development or shipping mode.
2. If you enabled automatic symbols upload, your build is ready to report crashes.
3. If you didn't enable automatic symbols upload, go to *Project Settings -> Ultimate Mobile Kit* and click *Upload dSYMs for Android [Development]* if your build was in development mode or *Upload dSYMs for Android [Shipping]* if your build was in shipping mode.
4. Please wait until uploading will finish.

ii. iOS

1. Go to *Project Settings -> iOS* and enable option *Generate dSYM file for third party crash tools*.
2. Build your game in development or shipping mode.
3. Go to *Project Settings -> Ultimate Mobile Kit* and click *Upload dSYMs for iOS*.
4. Firebase Crashlytics Dashboard will open.
5. Upload dSYM file which should be under the following path:
   a. Development: *[PROJECT_FOLDER]/Binaries/IOS/[PROJECT_NAME].dSYM.zip*
   b. Shipping: *[PROJECT_FOLDER]/Binaries/IOS/[PROJECT_NAME]-IOS-Shipping.dSYM.zip*
6. Please wait until uploading will finish.

16. Dynamic Links

Firebase Dynamic Links are links that work the way you want, on multiple platforms, and whether or not your game is already installed.

With Dynamic Links, your users get the best available experience for the platform they open your link on. If a user opens a Dynamic Link on iOS or Android, they can be taken directly to the linked content in your native game. If a user opens the same Dynamic Link in a desktop browser, they can be taken to the equivalent content on your website.

In addition, Dynamic Links work across game installs: if a user opens a Dynamic Link on iOS or Android and doesn't have your game installed, the user can be prompted to install it; then, after installation, your game starts and can access the link.

Official Firebase Dynamic Links documentation:
https://firebase.google.com/docs/dynamic-links/

Video introduction to Firebase Dynamic Links: https://youtu.be/LvY1JMcrPF8

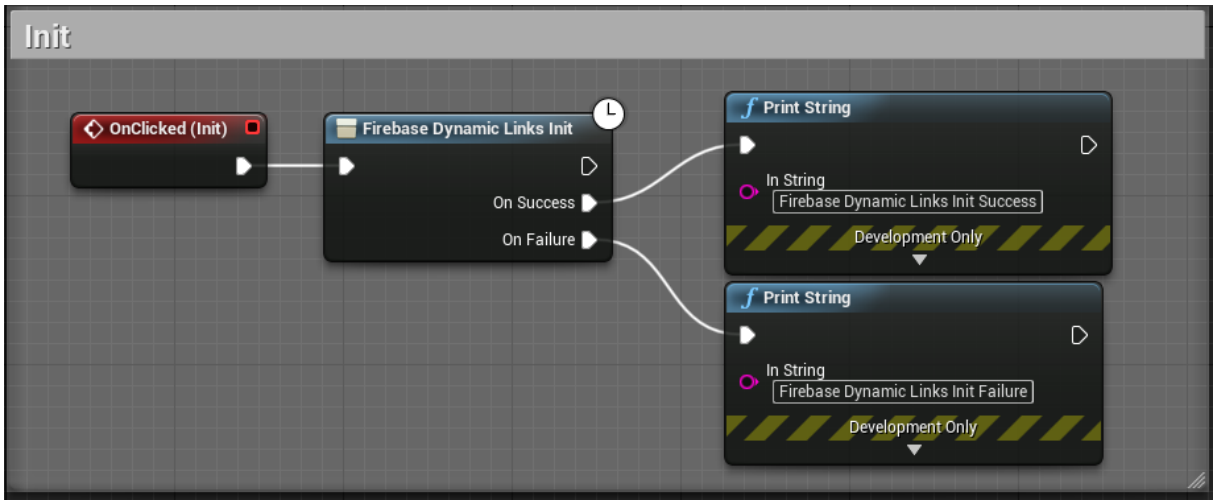
How does it work?

You create a Dynamic Link either by using the Firebase console, using a REST API, iOS or Android Builder API, or by forming a URL by adding Dynamic Link parameters to a domain specific to your game. These parameters specify the links you want to open, depending on the user's platform and whether your game is installed.

When a user opens one of your Dynamic Links, if your game isn't yet installed, the user is sent to the Google Play Store or App Store to install your game (unless you specify otherwise), and your game opens. You can then retrieve the link that was passed to your game and handle the link as appropriate for your game.

a. Initialization

Before making any call to the Firebase Dynamic Links you should first execute *Firebase Dynamic Links Init* function. If the result is a fail, study logs to find out what causes an issue.



b. Create Dynamic Links

There are three ways you can create a Dynamic Link:

- Using the Firebase console. This is useful if you're creating one-off links to share on social media.
- Using plugin's runtime functions on iOS and Android. This is the preferred way to dynamically create links in your game for user-to-user sharing or in any situation that requires many links. You can track the performance of Dynamic Links created with the runtime functions using Firebase Analytics.
- Manually. If you don't need to track click data and you don't care if the links are long, you can manually construct Dynamic Links using URL parameters, and by doing so, avoid an extra network round trip.

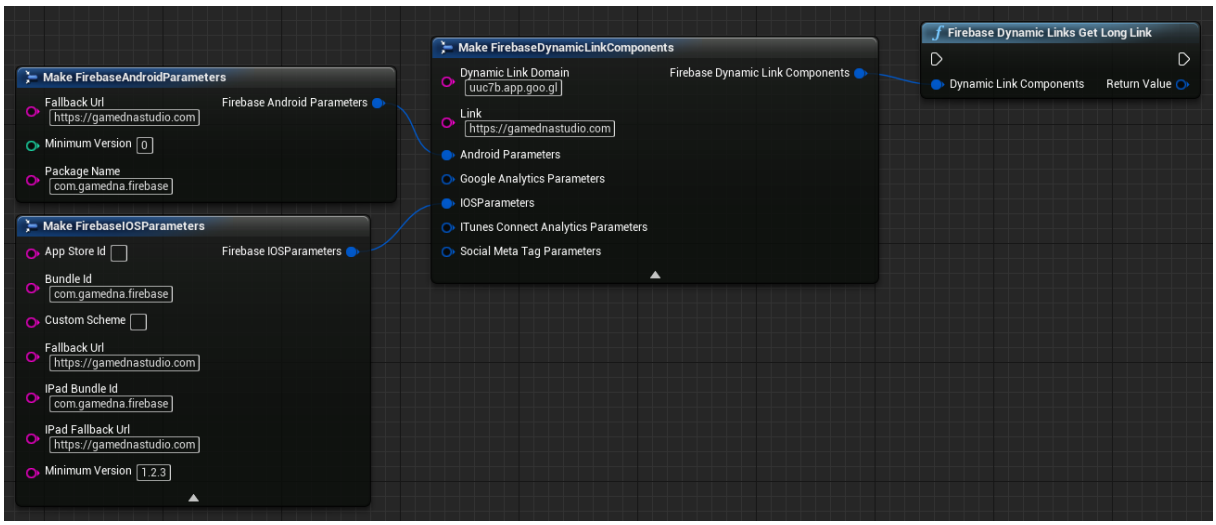
You can create short or long Dynamic Links using Ultimate Mobile Kit plugin. The API takes several optional parameter structures to build links. Short links can also be created from a previously generated long link. Firebase Dynamic Links generates a URL like the following:

```
https://abc123.app.goo.gl/WXYZ
```

i. Create a long Dynamic Link

To create a Dynamic Link, create a *Firebase Dynamic Link Components* structure, setting any of the optional members for additional configuration, and passing it to *Firebase Dynamic Link Get Short Link* or *Firebase Dynamic Link Get Long Link*.

The following minimal example creates a long Dynamic Link to https://gamednastudio.com that opens with your Android game *com.gamedna.firebase* and iOS game *com.gamedna.firebase*:

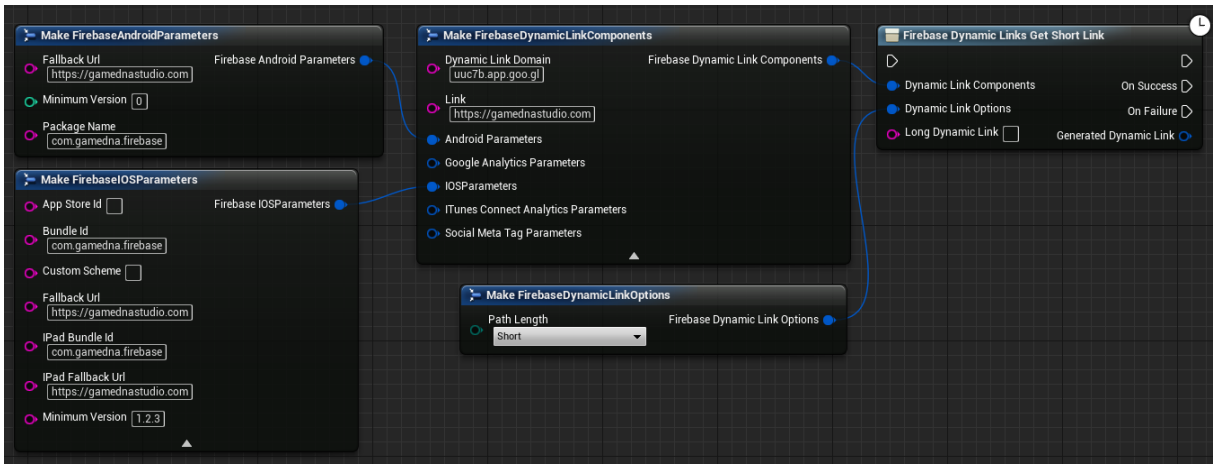


Long Links append all of the configuration settings as query arguments to the link and therefore do not require any network calls.

ii. Create a short Dynamic Link

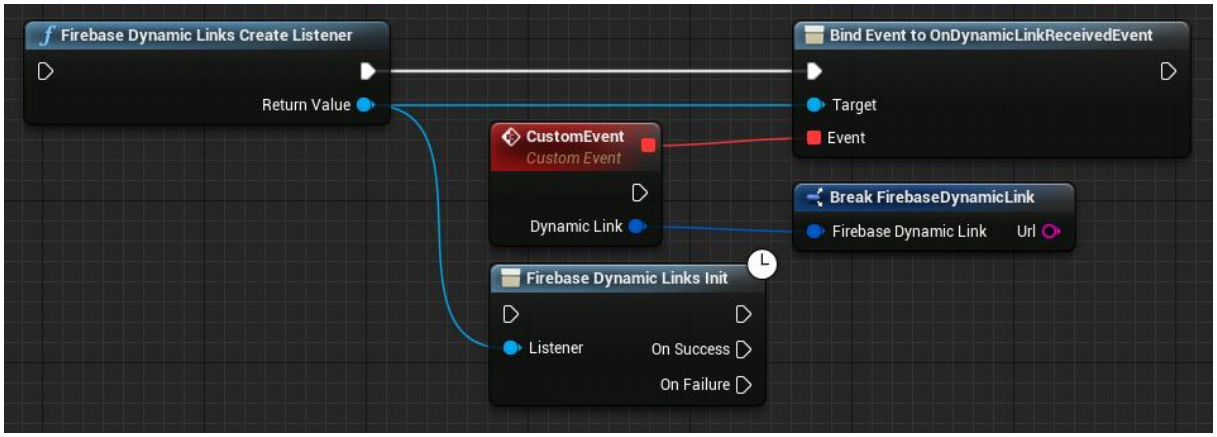
To create a short Dynamic Link, pass a previously generated long link to *Firebase Dynamic Links Get Short* Link or build *Firebase Dynamic Link Components* the same way as above.

Firebase Dynamic Links Get Short Link optionally takes an extra *Firebase Dynamic Link Options* config parameter with *Path Length*; this allows you to control how the link should be generated. Generating a short link requires a network request to the Firebase backend, so *Firebase Dynamic Links Get Short Link* is asynchronous.

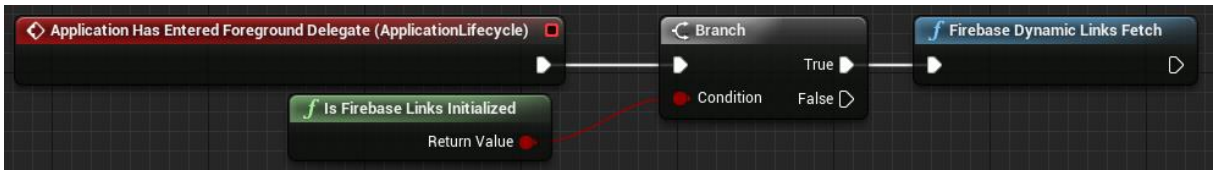


c. Receive Dynamic Links

To check for a received Dynamic Link, call *Firebase Dynamic Links Create Listener* function, bind event *OnDynamicLinkReceivedEvent*, and connect created Listener to *Firebase Dynamic Links Init*.



You must also call *Firebase Dynamic Links Fetch* when the application runs or gains focus in order for the listener to be triggered.



17. Invites

Firebase Invites are an out-of-the-box solution for game referrals and sharing via email or SMS.

Word of mouth is one of the most effective ways of getting users to install your game. In a recent study of thousands of smartphone users, researchers found that the #1 reason people discovered a game is because they heard about it from a friend or colleague. Firebase Invites makes it easy to turn your game's users into your game's strongest advocates.

Firebase Invites builds on Firebase Dynamic Links, which ensures that recipients of links have the best possible experience for their platform and the apps they have installed.

When a user taps one of your game's Share buttons and chooses the Firebase Invites channel—usually named "Email and SMS"—the Firebase Invites sharing screen opens. From the sharing screen, the user selects recipients from their Google contacts and contacts stored locally on the device, optionally customizes the invitation message and sends the invitations. Invitations are sent by email or SMS, depending on the available contact information, and contain a Dynamic Link to your app.

When the invitation's recipients open the Dynamic Link in the invitation, they are sent to the Google Play Store or App Store if they need to install your game; then, your game opens and can retrieve and handle the link.

Key capabilities:

- **Rich sharing that's easy for users** - Firebase Invites makes it simple for users to send content to their friends, over both SMS and email, by ensuring that referral codes, recipe entries, or other shared content gets passed along with the invitation—no cutting-and-pasting required.
- **Rich sharing that's easy to implement** - Firebase Invites handles the invitation flow for you, allowing you to deliver a straightforward user experience without taking engineering time away from the rest of your game.
- **Invitations that survive the installation process** - because Firebase Invites is built on Dynamic Links, invitations work across the App Store and Google Play Store installation processes and ensure that recipients get the referral code or shared content, whether or not they have your app installed.

Sending invitations:

- **Combines the most common sharing channels** - Firebase Invites can be sent over SMS or email.

- **Merged contacts selector** - the share screen's contact list is populated from the user's Google Contacts and the contacts stored locally on the device.
- **Recipient recommendations** - the share screen recommends recipients based on the contacts the user communicates with frequently.
- **Customizable invitation message** - you can set the default message to be sent with invitations. This message can be edited by the user when sending invitations.
- **Customizable rich-text email invitations** - you can customize email invitations in either of two ways:
    - Provide custom images that will be used along with additional text and graphics from the app's entry in the App Store or Play Store.
    - Provide HTML for a fully customized email invitation.
- **Low friction for users** - Android users can send invitations without signing in to your game.

Receiving invitations:

- **Installation flow initiation** - Firebase Invites smartly directs the recipient to the appropriate store when they open the link and need to install the game. iOS users are sent to the App Store, Android users are sent to the Google Play Store, and web users are sent to the store for the sender's platform.
- **Installation flow survival** - Invitations use Dynamic Links, which ensure that the link information contained in the invitation doesn't get lost, even if the user has to install the game first.
- **Low friction for users** - iOS and Android users can receive invitations without signing in to their Google Accounts.

Official Firebase Invites documentation: https://firebase.google.com/docs/invites/

Video introduction to Firebase Invites: https://youtu.be/LkaIJCZ_HyM

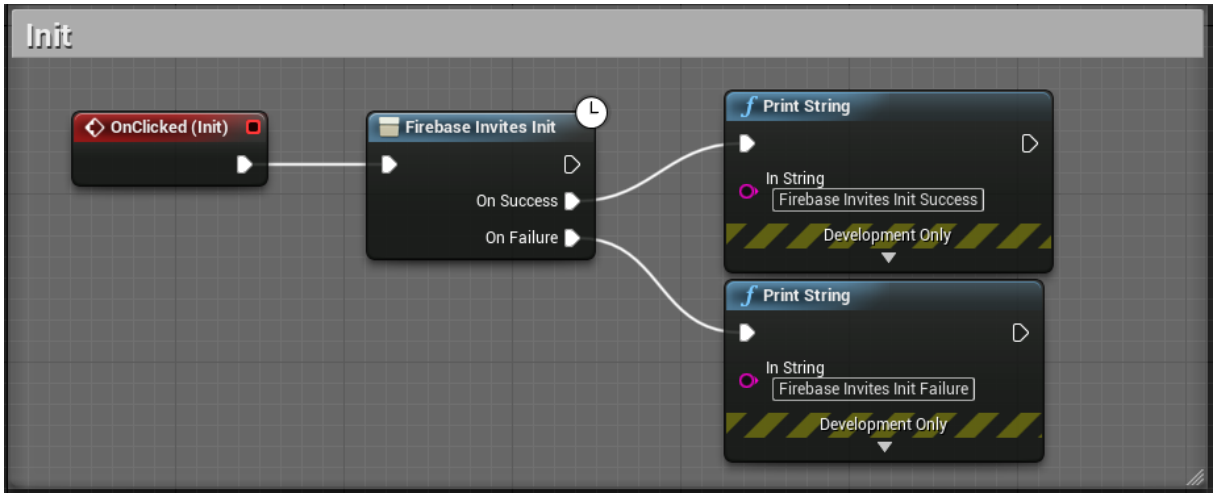
How does it work?

When a user taps one of your game's *Share* buttons and chooses the *Firebase Invites* channel - usually named *Email and SMS* - the Firebase Invites sharing screen opens. From the sharing screen, the user selects recipients from their Google contacts and contacts stored locally on the device, optionally customizes the invitation message and sends the invitations. Invitations are sent by email or SMS, depending on the available contact information, and contain a Dynamic Link to your game.

When the invitation's recipients open the Dynamic Link in the invitation, they are sent to the Play Store or App Store if they need to install your game; then, your game opens and can retrieve and handle the link.

a. Initialization

Before making any call to the Firebase Invites you should first execute *Firebase Invites Init* function. If the result is a fail, study logs to find out what causes an issue.



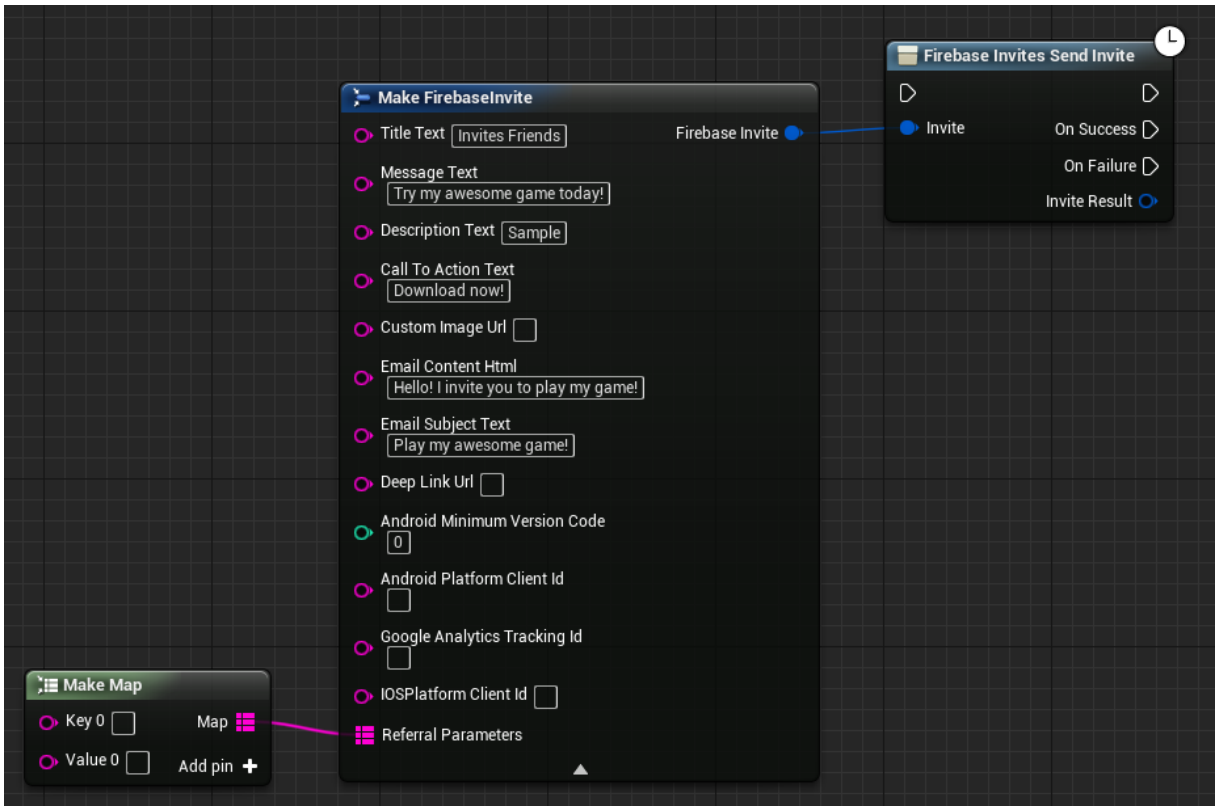
b. Send Invitation

Create a new instance of structure *Firebase Invite*. When you are ready to display the invitation call *Firebase Invites Send Invite* and pass previously created structure.

This will display the Invites client UI, allowing the user to choose the recipients and modify the message if desired. This UI will stay on the screen until the user chooses to either send the invitation or cancel.

If your project in the Firebase console contains exactly one application for each platform, Firebase Invites will automatically associate the applications with each other, so that (for example) iOS users clicking on an invitation sent by an Android user will be sent to the right place to install your game on their platform.

If you have more than one application on each platform, you can set *iOS Platform Client Id* or *Android Platform Client Id*, passing in the Firebase client ID of the alternate platform.



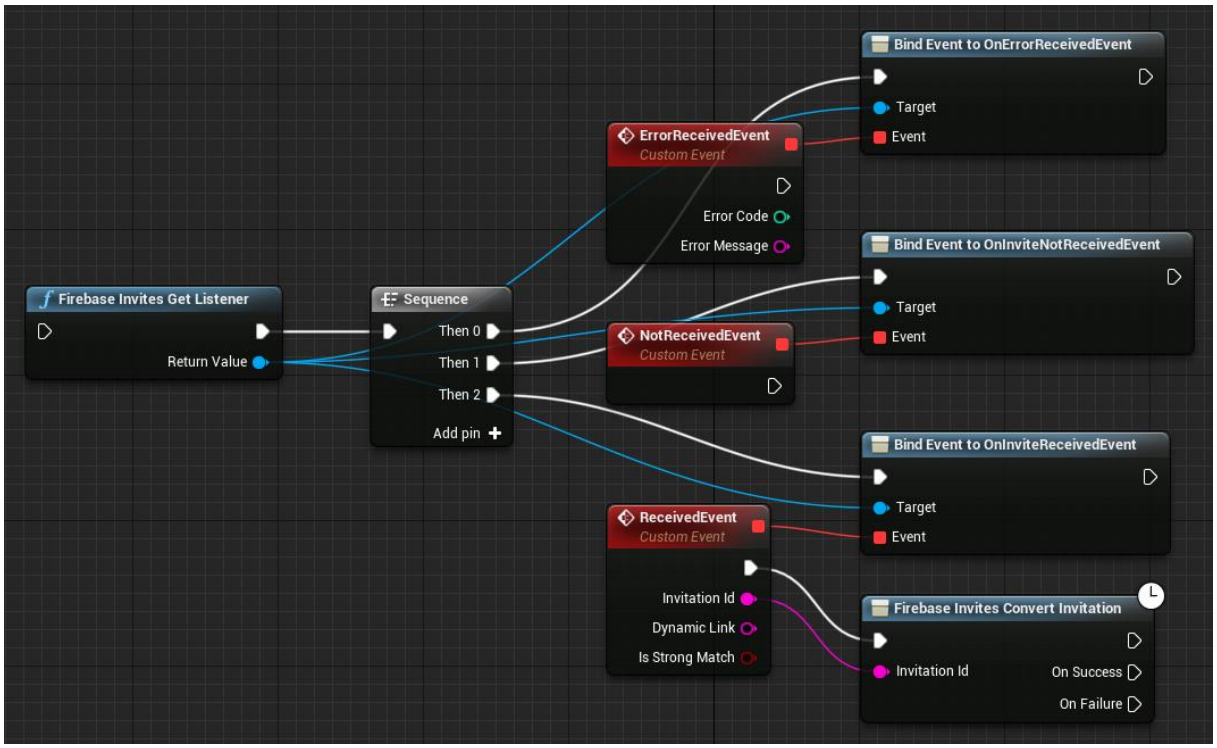
c. Receive Invitation

When a user receives an invitation, if the user has not yet installed the game, they can choose to install the game from their platform's app store.

Then, after the game is installed, or if the game was already installed, the game starts and receives the URL to its content, if you sent one. The game will also receive the invitation ID, which will match the invitation ID on the sending side.

To check for a received Invitation, call *Firebase Invites Get Listener* function and bind event *OnInviteReceivedEvent*, *OnInviteNotReceivedEvent*, and *OnErrorReceivedEvent*.

When your game receives new Invitation and calls *OnInviteReceivedEvent*, you should also call *Firebase Invites Convert Invitation* to mark the Invitation as converted in app-specific way.


Bind Event to OnErrorReceivedEvent
Target
Event
ErrorReceivedEvent
Custom Event
Error Code
Error Message
Bind Event to OnInviteNotReceivedEvent
Target
Event
Firebase Invites Get Listener
Return Value
Sequence
Then 0
Then 1
Then 2
Add pin
NotReceivedEvent
Custom Event
Bind Event to OnInviteReceivedEvent
Target
Event
ReceivedEvent
Custom Event
Invitation Id
Dynamic Link
Is Strong Match
Firebase Invites Convert Invitation
Invitation Id
On Success
On Failure

18. In-App Messaging

Firebase In-App Messaging helps you engage your app's active users by sending them targeted, contextual messages that encourage them to use key app features. For example, you could send an in-app message to get users to subscribe, watch a video, complete a level, or buy an item. You can customize messages as banners, modals, or images, and set up triggers so that they appear exactly when they'd benefit your users most.

Use Firebase In-App Messaging to encourage exploration and discovery: highlight a sale or coupon in your ecommerce app, give clues or tips in your game, or prompt a like or share in your social media app.

Key capabilities:

- **Send relevant, engaging messages** - Firebase In-App Messaging sends messages when they're most needed: while users are actually in your app. Promote your big sale when users visit your in-app store, not while they're in line at the grocery store. Highlight that cool, new level when users play your game, not when they're sitting down to watch the big game.
- **Target messages by audience or behavior** - Firebase In-App Messaging works with Analytics and Predictions to give you tools to deliver messages to the users you'd most like to reach. Send messages based on users' demographics, past behavior, or even predictions of their future behavior.
- **Create flexible, custom alerts** - with the ability to customize your messages' style, appearance, display triggers, and content all in a few clicks, Firebase In-App Messaging helps you do everything from sending promotional offers to getting users to update to a new version of your app.

Support for Firebase In-App Messaging is already enabled in the Ultimate Mobile Kit plugin.

Official Firebase In-App Messaging documentation:
https://firebase.google.com/docs/in-app-messaging/

Video introduction to Firebase In-App Messaging: https://youtu.be/5MRKpvKV2pg

19. Test Lab

Firebase Test Lab is a cloud-based app-testing infrastructure. With one operation, you can test your Android or iOS app across a wide variety of devices and device configurations, and see the results - including logs, videos, and screenshots - in the Firebase console. Test Lab uses real, production devices running in a Google data center to test your app. The devices are flashed with updated APIs and have customizable locale settings, allowing you to road-test your app on the hardware and configurations it'll encounter in real-world use.

Key capabilities:

- **Test Android and iOS apps** - if your app has both an Android and iOS version, no worries. Test Lab now offers iOS devices to test on.
- **Run on real devices** - Test Lab exercises your app on devices installed and running in a Google data center, so you can find issues that only occur on specific devices and configurations.
- **Workflow integration** - Test Lab is integrated with the Firebase console, Android Studio, and the gcloud command line tool. You can even use it with Continuous Integration (CI) systems.

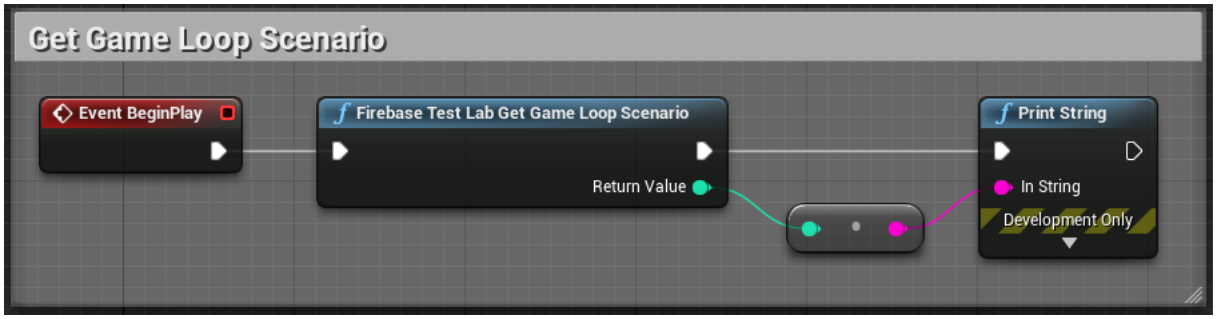
Official Firebase Test Lab documentation: https://firebase.google.com/docs/test-lab/

Video introduction to Firebase Test Lab: https://youtu.be/4_ZEEX1x17k

Automating game testing is challenging because of the wide range of UI frameworks used for game development (some of which are engine-dependent), and the difficulties of automating UI navigation in games. To support game app testing, Test Lab now includes support for using a "demo mode" where the game app runs while simulating the actions of a player. This mode can include multiple loops (or scenarios).

To use the game loop test in Test Lab, your game must be modified to do the following:

1. Launch the loop
2. Run the loop
3. Close the game app



You should call function *Firebase Test Lab Get Game Loop Scenario* when your game is launched (for example in *Event BeginPlay* in the main level). You can then implement your game loop (or multiple loops) based on provided *Game Loop Scenario*. For example, game loops can be used to:

- Run a level of your game the same way an end user would play it. You can either script the input of the player, let the player idle, or replace the user with an AI if it makes sense in your game (for example, if you already have an AI implemented, like in a car racing game, and you can easily put an AI driver in charge of the player's input).
- Run your game at the highest quality setting to see if devices support it.
- Run a technical test (compile multiple shaders, execute them, check that the output is as expected, etc).

At any point in your test where you want to take a screenshot, call the *Firebase Test Lab Take Screenshot* method, where the argument is a label that you use to identify the screenshot (*TestLabel* is used in the example). After your test has completed, you can compare any screenshots taken during testing by selecting an element in the test results tree and then clicking the *View Screenshots* option.

You can also support multiple game loops in your game app. For example, if you have multiple levels in your game, you might want to have one game loop to launch each level instead of having one loop that iterates through all of them.

That way, if your game crashes on level 32 you can directly launch that game loop to try and reproduce the crash and to test bug fixes.

For example, if you have 5 game loops in your game go to *Project Settings -> Ultimate Mobile Kit -> Firebase Test Lab* and set *Number Of Game Loops* to 5.

Then, when you launch *Test Lab Test Loop Manager*, you can select which loop to launch. If you select multiple loops to launch, it launches each loop in sequence after the preceding loop completes. In each loop you can call *Firebase Test Lab Get Game Loop Scenario* and get current *Scenario (Loop Number)* in order to make actions based on this. *Scenario* parameter is in the range of 1 to the maximum number of loops supported.

At the end of your test you should close the game using *Firebase Test Lab Finish Test*.

Generally, the game loop test allows the UI framework to start the next loop. If you do not close the game, the UI framework running your loops won't know that the test has finished, and may terminate the game after some time has passed.

20. Cloud Functions

Cloud Functions for Firebase lets you automatically run backend code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers.

The functions you write can respond to events generated by these other Firebase and Google Cloud features:

- Firebase Authentication Triggers
- Firebase Analytics Triggers
- Firebase Crashlytics Triggers
- Firebase Cloud Storage Triggers
- Firebase Realtime Database Triggers
- Firebase Remote Config Triggers
- Cloud Pub/Sub Triggers
- HTTP Triggers
- Cloud Firestore Triggers

Support for Firebase Cloud Functions in the Ultimate Mobile Kit includes only server side. For calling Cloud Functions directly from your game you need to have the Realtime Database plugin: https://www.gamednastudio.com/realtime-database/

Official Firebase Cloud Functions documentation:
https://firebase.google.com/docs/functions/

Video introduction to Firebase Cloud Functions: https://youtu.be/vr0Gfvp5v1A

Deploy your JavaScript or TypeScript code to Google servers with one command from the command line. After that, Firebase automatically scales up computing resources to match the usage patterns of your users. You never worry about credentials, server configuration, provisioning new servers, or decommissioning old ones.

In many cases, developers prefer to control application logic on the server to avoid tampering on the client side. Also, sometimes it's not desirable to allow that code to be reverse engineered. Cloud Functions is fully insulated from the client, so you can be sure it is private and always does exactly what you want.

How does it work?

After you write and deploy a function, Google's servers begin to manage the function immediately, listening for events and running the function when it is triggered. As the load increases or decreases, Google responds by rapidly scaling the number of virtual server instances needed to run your function.

Lifecycle of a function

1. The developer writes code for a new function, selecting an event provider (such as Realtime Database), and defining the conditions under which the function should execute.
2. The developer deploys the function, and Firebase connects it to the selected event provider.
3. When the event provider generates an event that matches the function's conditions, the code is invoked.
4. If the function is busy handling many events, Google creates more instances to handle work faster. If the function is idle, instances are cleaned up.
5. When the developer updates the function by deploying updated code, all instances for the old version are cleaned up and replaced by new instances.
6. When a developer deletes the function, all instances are cleaned up, and the connection between the function and the event provider is removed.

What Can I Do with Firebase Cloud Functions?

1. Notify players when something interesting happens
    a. Send message to the player when he completes for example 10 achievements.
    b. Send confirmation emails to players subscribing/unsubscribing to a newsletter.
    c. Send a welcome email when a user completes tutorial.
    d. Send an SMS confirmation when player creates a new account.
2. Perform Realtime Database sanitization and maintenance
    a. Convert text to emoji.
    b. Purge a deleted user’s content from Realtime Database
    c. Limit the number of child nodes in a Firebase database.
    d. Track the number of elements in a Realtime Database list.
    e. Copy data from Realtime Database to Google Cloud BigQuery.
    f. Manage computed metadata for database records.
3. Execute intensive tasks in the cloud instead of in your game

    a. Periodically delete unused Firebase accounts.
    b. Automatically moderate uploaded images.
    c. Send bulk email to users.
    d. Aggregate and summarize data periodically.
    e. Process a queue of pending work.
4. Integrate with third-party services and APIs
    a. Use Google Cloud Vision API to analyze and tag uploaded images.
    b. Translate messages using Google Translate.
    c. Use auth providers like LinkedIn or Instagram to sign in users.
    d. Send a request to a webhook on Realtime Database writes.
    e. Enable full-text search on Realtime Database elements.
    f. Process payments from users.
    g. Create auto-responses to phone calls and SMS messages.
    h. Create a chatbot using Google Assistant.
5. And much more… practically infinite possibilities…

21. Predictions

Firebase Predictions applies machine learning to your analytics data to create dynamic user groups based on your players' predicted behavior. These predictions are automatically available for use with Firebase Remote Config, the Notifications composer, and A/B testing.

When you use Predictions with Remote Config, you can increase conversions by providing a custom experience based on each of your users' predicted behavior.

You can use Predictions with the Notifications composer to deliver the right message to the right user groups.

And, with A/B testing, you can evaluate the effectiveness of your prediction-based strategies.

Support for Firebase Predictions is already enabled in the Ultimate Mobile Kit plugin.

Official Firebase Predictions documentation:
https://firebase.google.com/docs/predictions/

Video introduction to Firebase Predictions: https://youtu.be/ORrvrVEHJz4

Key capabilities:

- **Bring the power of Google's machine learning to your data** – Firebase Predictions applies Google's expertise in machine learning to your analytics data, creating dynamic user groups based on users' predicted behavior. With Google's powerful machine learning, you can make product decisions based on predicted behavior, rather than historic behavior.
- **Boost conversions through dynamic, customized user experiences** – Firebase Predictions is integrated with Remote Config, letting you customize a player's experience based on their predicted behavior. For example, for players who are predicted to spend, you can show a new in-app purchase bundle, while for players who are predicted to not spend, you can adjust the frequency of ads. Groups update dynamically over time as an individual's prediction changes, so you can always offer a fresh, personalized experience to your players.
- **Increase retention with smarter notifications** – re-engaging a player who has already stopped using your game is tough. By using Predictions, you can engage players who are predicted to churn, before they ever stop using your game.

- **Create custom predictions** – in addition to the built-in predictions - will churn, will not churn, will spend, and will not spend - Firebase Predictions allows you to create predictions based on any conversion event in your analytics data. Once you define the event, Predictions creates a dynamic user group composed of players who are predicted to complete that event in your game.

How does it work?

Predictions are available for iOS and Android games that include the Firebase Analytics SDK. Predictions creates dynamic groups of players who are likely to complete a certain event over the next 7 days. You can use these groups to target users with Remote Config and the Notifications composer.

By default, Predictions provides two types of predictions: churn, which predicts which users will disengage from your game over the next 7 days (that is, they will not open the game or game-related notification messages), and spend, which predicts which players will spend money in your game over the next seven days. You can also create your own predictions based on custom conversion Analytics events that you collect in your game.

Predictions lets you adjust the risk tolerance of a prediction so that you can strike the right balance between targeting fewer users with more accuracy, or more users with less accuracy, and it shows you what percentage of your user base will be targeted at each risk tolerance level. The machine learning model for your game improves as the amount and relevance of data collected using Analytics increases, and as your number of users increases. In addition, the accuracy of the model for a specific user will improve further after that user has used the game for at least a few days.

Predictions creates groups of players, called predicted user groups. You can send notifications to the predicted user groups. Or, you can change the behavior or appearance of app instances used by predicted user groups using Remote Config.

22. A/B Testing

When you are updating your game and using Firebase Remote Config to push it to an game with an active user base, you want to make sure you get it right. You might be uncertain about the following:

- **The best way to implement a feature to optimize the user experience.** Too often, game developers don't learn that their players dislike a new feature or an updated user experience until their game's rating in the app store declines. A/B testing can help measure whether your players like new variants of features, or whether they prefer the game as it currently exists. Plus, keeping most of your players in a control group ensures that most of your user base can continue to use your game without experiencing any changes to its behavior or appearance until the experiment has concluded.
- **The best way to optimize the user experience for a business goal.** Sometimes you're implementing product changes to maximize a metric like revenue or retention. With A/B testing, you set your business objective, and Firebase does the statistical analysis to determine if a variant is outperforming the control group for your selected objective.

To A/B test feature variants with a control group, do the following:

1. Create your experiment.
2. Validate your experiment on a test device.
3. Manage your experiment.

Support for Firebase A/B Testing is already enabled in the Ultimate Mobile Kit plugin.

Official Firebase A/B Testing documentation:
https://firebase.google.com/docs/remote-config/abtest-config

Video introduction to Firebase A/B Testing: https://youtu.be/ROdfwKFH6u8

23. Hosting

Firebase Hosting provides fast and secure static hosting for your web app.

Firebase Hosting is production-grade web content hosting for developers. With Hosting, you can quickly and easily deploy web apps and static content to a global content-delivery network (CDN) with a single command.

Support for Firebase Hosting is already enabled in the Ultimate Mobile Kit plugin.

Official Firebase Hosting documentation: https://firebase.google.com/docs/hosting/

Video introduction to Firebase Hosting: https://youtu.be/jsRVHeQd5kU

Key capabilities:

- **Served over a secure connection** – the modern web is secure. Zero-configuration SSL is built into Firebase Hosting so content is always delivered securely.
- **Fast content delivery** – each file you upload is cached on SSDs at CDN edges around the world. No matter where your users are, the content is delivered fast.
- **Rapid deployment** – using the Firebase CLI, you can get your app up and running in seconds. Command line tools make it easy to add deployment targets into your build process.
- **One-click rollbacks** – quick deployments are great, but being able to undo mistakes is even better. Firebase Hosting provides full versioning and release management with one-click rollbacks.

How does it work?

Firebase Hosting is built for the modern web developer. Static sites are more powerful than ever with the rise of front-end JavaScript frameworks like Angular and static generator tools like Jekyll. Whether you are deploying a simple app landing page or a complex Progressive Web App, Hosting gives you the infrastructure, features, and tooling tailored to deploying and managing static websites.

Hosting gives your project a subdomain on the firebaseapp.com domain. Using the Firebase CLI, you can deploy files from local directories on your computer to your Hosting server. Files are served over an SSL connection from the closest edge server on Google's global CDN.

In addition to static content hosting, Firebase Hosting offers lightweight configuration options for you to be able to build sophisticated Progressive Web Apps. You can easily rewrite URLs for client-side routing or set up custom headers.

Once you're ready to take a site to production, you can connect your own domain name to Firebase Hosting. Firebase automatically provisions an SSL certificate for your domain so all your content is served securely.

24. C++ implementation

Add the following line to your **.Build.cs* file private dependency module name:

```
PrivateDependencyModuleNames.AddRange(new string[] { "UltimateMobileKit"
});
```

Before you call needed plugin function you should add necessary header:

```
#include "UltimateMobileKit.h"
```

and init proper Firebase module somewhere (in this case this is Firebase Analytics):

```
void TestGameMode::InitFirebase()
{
    // Retrieving Ultimate Mobile Kit subsystem
    FUltimateMobileKit* UltimateMobileKit = FUltimateMobileKit::Get();

    // Checking if Ultimate Mobile Kit subsystem is valid
    if (UltimateMobileKit)
    {
        // Retrieving Firebase Analytics module (we recommend using Shared
Pointers)
        TSharedPtr<FFirebaseAnalytics, ESPMode::ThreadSafe>
FirebaseAnalytics = UltimateMobileKit->GetFirebaseAnalytics();

        // Checking if Firebase Analytics module is valid
        if (FirebaseAnalytics.IsValid())
        {
            // Creating delegate for Init function
            FOnFirebaseAnalyticsInitializeCompleteDelegate
InitializeCompleteDelegate =

FOnFirebaseAnalyticsInitializeCompleteDelegate::CreateUObject(this,
&ThisClass::OnInitCompleted);

            // Calling Firebase Analytics Init function
            FirebaseAnalytics->Init(InitializeCompleteDelegate);

            return;
        }
    }
}

void TestGameMode::OnInitCompleted(bool bSuccess)
{
}
```

You have now access to all functions from initialized Firebase module. For example *Set User Id*:

```
FirebaseAnalytics->SetUserId(TEXT("User_Id"));
```

or *Log Event* with multiple parameters:

```
TArray<UFirebaseVariant*> Parameters = TArray<UFirebaseVariant*>();
UFirebaseVariant* Parameter1 =
UFirebaseVariant::FirebaseStringVariant(TEXT("Parameter1_Name"),
TEXT("Parameter1_Value"));
```

```
UFirebaseVariant* Parameter2 =
UFirebaseVariant::FirebaseBooleanVariant(TEXT("Parameter2_Name"), true);
UFirebaseVariant* Parameter3 =
UFirebaseVariant::FirebaseIntegerVariant(TEXT("Parameter3_Name"), 28);
UFirebaseVariant* Parameter4 =
UFirebaseVariant::FirebaseFloatVariant(TEXT("Parameter4_Name"), 67.12f);

FirebaseAnalytics->LogEvent(TEXT("Test_Event"), Parameters);
```