Android | iOS | macOS | Web | Windows | Electron | Flutter

React Native | Python | React JS | Unity | Unreal Engine | Unreal (Blueprint)

Android

On this page

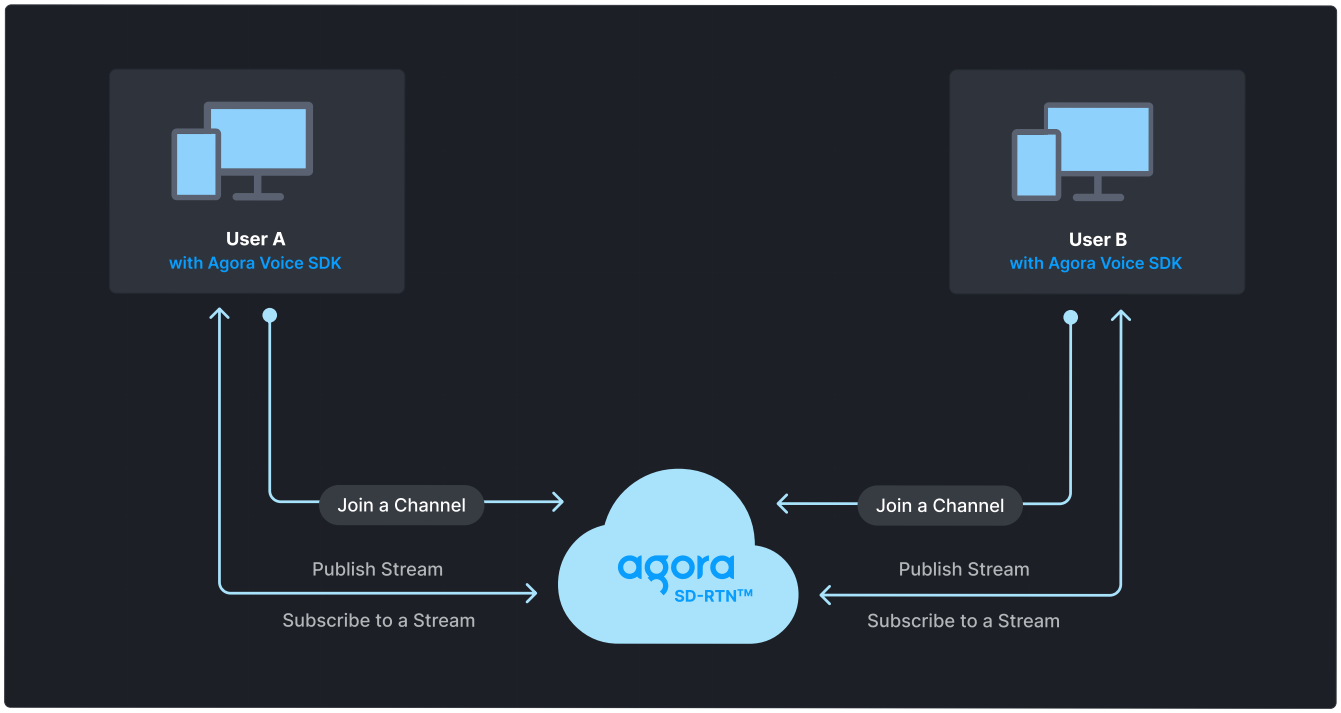
Quickstart

This page provides a step-by-step guide on how to create a basic Voice Calling app using the Agora Voice SDK.

Understand the tech

To start a Voice Calling session, implement the following steps in your app:

- **Initialize the Agora Engine**: Before calling other APIs, create and initialize an Agora Engine instance.
- **Join a channel**: Call methods to create and join a channel.
- **Send and receive audio**: All users can publish streams to the channel and subscribe to audio streams published by other users in the channel.



Prerequisites

- Android Studio 4.2 or higher.
- Android SDK API Level 21 or higher.
- Two mobile devices running Android 5.0 or higher.
- A microphone
- A valid Agora account and project. Please refer to Agora account management for details.

Set up your project

This section shows you how to set up your Android project and install the Agora Voice SDK.

Create a new project | **Add to an existing project**

1. Create a new project.
   i. Open Android Studio and select **File > New > New Project...**.
   ii. Select **Phone and Tablet** > **Empty Activity** and click **Next**.
   iii. Set the project name and storage path.
   iv. Select **Java** or **Kotlin** as the language, and click **Finish** to create the Android project.

> **NOTE**
> After you create a project, Android Studio automatically starts gradle sync. Ensure that the synchronization is successful before proceeding to the next step.

2. Add a layout file for your activity.

   Set up a basic layout for your activity. Refer to Create a user interface to get a bare bones sample layout.

Install the SDK

Use either of the following methods to add Voice SDK to your project.

Maven Central | **Manual integration**

1. Open the `settings.gradle` file in the project's root directory and add the Maven Central dependency, if it doesn't already exist:

```
repositories {
    mavenCentral()
}
```

> **INFO**

> If your Android project uses **dependencyResolutionManagement**, the method of adding the Maven Central dependency may differ.

2. To integrate the Voice SDK into your Android project, add the following to the `dependencies` block in your project module `build.gradle` file:

   - Groovy `build.gradle`

     ```
     implementation 'io.agora.rtc:voice-sdk:x.y.z'
     ```

   - Kotlin `build.gradle.kts`

     ```
     implementation("io.agora.rtc:voice-sdk:x.y.z")
     ```

   Replace `x.y.z` with the specific SDK version number, such as `4.5.0`.

   > **INFO**
   >
   > To get the latest version number, check the **Release notes**. To integrate the Lite SDK, use `io.agora.rtc:lite-sdk` instead.

3. Prevent code obfuscation

   Open the `/app/proguard-rules.pro` file and add the following lines to prevent the Voice SDK code from being obfuscated:

   ```
   -keep class io.agora.** { *; }
   -dontwarn io.agora.**
   ```

Implement Voice Calling

This section guides you through the implementation of basic real-time audio interaction in your app.

The following figure illustrates the essential steps:

▶ Quick start sequence

This guide includes **complete sample code** that demonstrates implementing basic real-time interaction. To understand the core API calls in the sample code, review the following implementation steps and use the code in your `MainActivity` file.

Import Agora classes

Import the relevant Agora classes and interfaces:

Java **Kotlin**

```
import io.agora.rtc2.ChannelMediaOptions
import io.agora.rtc2.Constants
import io.agora.rtc2.IRtcEngineEventHandler
import io.agora.rtc2.RtcEngine
import io.agora.rtc2.RtcEngineConfig
```

Initialize the engine

For real-time communication, initialize an `RtcEngine` instance and set up event handlers to manage user interactions within the channel. Use `RtcEngineConfig` to specify the application context, **App ID**, and custom **event handler**, then call `RtcEngine.create(config)` to initialize the engine, enabling further channel operations. In your `MainActivity` file, add the following code:

Java **Kotlin**

```
// Fill in the App ID obtained from the Agora Console
private val myAppId = "<Your app ID>"
private var mRtcEngine: RtcEngine? = null

private fun initializeAgoraVoiceSDK() {
    try {
        val config = RtcEngineConfig().apply {
            mContext = baseContext
            mAppId = myAppId
            mEventHandler = mRtcEventHandler

        }

        mRtcEngine = RtcEngine.create(config)

    } catch (e: Exception) {

        throw RuntimeException("Error initializing RTC engine: ${e.message}")

    }

}

```

Join a channel

To join a channel, call `joinChannel` with the following parameters:

- **Channel name**: The name of the channel to join. Clients that pass the same channel name join the same channel. If a channel with the specified name does not exist, it is created when the first user joins.

- **Authentication token**: A dynamic key that authenticates a user when the client joins a channel. In a production environment, you obtain a token from a **token server** in your security infrastructure. For the purpose of this guide **Generate a temporary token**.
- **User ID**: A 32-bit signed integer that identifies a user in the channel. You can specify a unique user ID for each user yourself. If you set the user ID to `0` when joining a channel, the SDK generates a random number for the user ID and returns the value in the `onJoinChannelSuccess` callback.
- **Channel media options**: Configure `ChannelMediaOptions` to define publishing and subscription settings, optimize performance for your specific use-case, and set optional parameters.

For Voice Calling, set the `channelProfile` to `CHANNEL_PROFILE_COMMUNICATION` and the user role to `CLIENT_ROLE_BROADCASTER`.

Java Kotlin

```
// Fill in the channel name
private val channelName = "<Your channel name>"
// Fill in the temporary token generated from Agora Console
private val token = "<Your token>"

private fun joinChannel() {
    val options = ChannelMediaOptions().apply {
        clientRoleType = Constants.CLIENT_ROLE_BROADCASTER
        channelProfile = Constants.CHANNEL_PROFILE_COMMUNICATION
        publishMicrophoneTrack = true;

    }

    mRtcEngine?.joinChannel(token, channelName, 0, options)

}
```

Subscribe to Voice SDK events

The Voice SDK provides an interface for subscribing to channel events. To use it, create an instance of `IRtcEngineEventHandler` and implement the event methods you want to handle.

> **INFO**
>
> To ensure that you receive all Voice SDK events, set the Agora Engine event handler before joining a channel.

Java Kotlin

```
private val mRtcEventHandler = object : IRtcEngineEventHandler() {
```

```
    override fun onJoinChannelSuccess(channel: String?, uid: Int, elapsed: Int) {
        super.onJoinChannelSuccess(channel, uid, elapsed)
        runOnUiThread {
            showToast("Joined channel $channel")
        }
    }
    override fun onUserJoined(uid: Int, elapsed: Int) {
        runOnUiThread {
            showToast("User joined: $uid")

        }

    }

    override fun onUserOffline(uid: Int, reason: Int) {

        super.onUserOffline(uid, reason)

        runOnUiThread {

            showToast("User offline: $uid")

        }

    }

}
```

Handle permissions

To access the microphone on Android devices, declare the necessary permissions in the app's manifest and ensure that the user grants these permissions when the app starts.

1. Open your project's `AndroidManifest.xml` file and add the following permissions before `<application>`:

```
<!--Required permissions-->
<uses-permission android:name="android.permission.INTERNET"/>

<!--Optional permissions-->
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<!-- For devices running Android 12 (API level 32) or higher and integrating Aac
```

```
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
<!-- For Android 12 or higher, the following permissions are also required -->
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>
```

2. Use the following code to handle runtime permissions in your Android app. The logic ensures that the necessary permissions are granted before starting Voice Calling. In your `MainActivity` file, add the following code:

Java **Kotlin**

```
private val permissionReqId = 22

 private fun checkPermissions(): Boolean {
     for (permission in getRequiredPermissions()) {
         if (ContextCompat.checkSelfPermission(this, permission) !=
PackageManager.PERMISSION_GRANTED) {
             return false
         }
     }
     return true
 }

 private fun requestPermissions() {

     ActivityCompat.requestPermissions(this, getRequiredPermissions(),
PERMISSION_REQ_ID)
 }

 private fun getRequiredPermissions(): Array<String> {

     return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {

         arrayOf(

             Manifest.permission.RECORD_AUDIO,

             Manifest.permission.READ_PHONE_STATE,

             Manifest.permission.BLUETOOTH_CONNECT

         )

     } else {

```

```
            arrayOf(Manifest.permission.RECORD_AUDIO)

        }

    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions:
Array<out String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)

        if (checkPermissions()) {

            startVoiceCalling()

        }

    }

```

Start and close the app

When a user launches your app, start real-time interaction. When a user closes the app, stop the interaction.

1. In the `onCreate` callback, check whether the app has been granted the required permissions. If the permissions have not been granted, request the required permissions from the user. If permissions are granted, initialize `RtcEngine` and join a channel.

Java **Kotlin**

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    if (checkPermissions()) {
        startVoiceCalling()
    } else {
        requestPermissions()
    }
}
```

2. When a user closes the app, or switches the app to the background, call `leaveChannel` to leave the current channel and release all session-related resources.

Java **Kotlin**

```
private fun cleanupAgoraEngine() {
```

```
        mRtcEngine?.apply {
            leaveChannel()
        }
        mRtcEngine = null
    }
```

Complete sample code

A complete code sample demonstrating the basic process of real-time interaction is provided for your reference. To use the sample code, copy the following lines into the `MainActivity` file in your project. Then, replace `<projectname>` in package `com.example.<projectname>` with your project's name.

▶ Complete sample code for real-time Voice Calling

> **INFO**
>
> For the `myAppId` and `token` variables, replace the placeholders with the values you obtained from Agora Console. Ensure you enter the same `channelName` you used when generating the temporary token.

Create a user interface

Use the following code to generate a basic user interface. Paste the code into the `/app/src/main/res/layout/activity_main.xml` file, replacing the existing content.

▼ Sample code to create the user interface

```
<?xml version="1.0" encoding="UTF-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/darker_gray"
    tools:context=".MainActivity">


    <TextView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

```

```
        android:layout_centerInParent="true"

        android:text="Welcome to Agora Voice Calling." />


</RelativeLayout>
```

Test the sample code

Take the following steps to test the sample code:

1. In `MainActivity` update the values for `myAppId`, and `token` with values from Agora Console. Fill in the same `channelName` you used to generate the token.
2. Enable developer options on your Android test device. Turn on USB debugging, connect the Android device to your development machine through a USB cable, and check that your device appears in the Android device options.
3. In Android Studio, click  **Sync Project with Gradle Files** to resolve project dependencies and update the configuration.
4. After synchronization is successful, click  **Run app**. Android Studio starts compilation. After a few moments, the app is installed on your Android device.
5. Launch the App, grant the recording permission.
6. On a second Android device, repeat the previous steps to install and launch the app. Alternatively, use the **Web demo** to join the same channel and test the following use-cases:
   - If users on both devices join the channel as hosts, they can hear each other.
   - If one user joins as host and the other as audience, the audience can hear the host.

Reference

This section contains content that completes the information on this page, or points you to documentation that explains other aspects to this product.

- If a firewall is deployed in your network environment, refer to **Connect with Cloud Proxy** to use Agora services normally.

Next steps

After implementing the quickstart sample, read the following documents to learn more:

- To ensure communication security in a test or production environment, best practice is to obtain and use a token from an authentication server. For details, see **Secure authentication with tokens**.

Sample project

Agora provides open source sample projects on GitHub for your reference. Download or view the JoinChannelAudio project for a more detailed example.

API reference

- `RtcEngineConfig`
- `create`
- `ChannelMediaOptions`
- `joinChannel`
- `leaveChannel`
- `IRtcEngineEventHandler`

Frequently asked questions

- How can I listen for audience joining or leaving a channel?
- How can I solve channel-related issues?
- How can I set the log file?
- Why do apps on some Android versions fail to capture audio and video after screen locking or switching to the background?

See also

- Error codes
- Connection status management

Suggest an edit