# [Audio Filter Plugins](https://confluence.agoralab.co/display/TEKP/Audio+Filter+Plugins)

[Audio Filter Plugins](#_83lsnix3czjw)

[Understand the tech](#_vlxdgn5utkxg)

[Prerequisites](#_c08qgwsaw0jf)

[Project setup](#_85bpa5yl25gv)

[Build an audio filter plugin](#_vknlt1wuifd1)

[Implement IAudioFilter](#_i7bbqi1g3x7y)

[Encapsulate the audio filter plugin](#_9kssnu4qfubv)

[Package the plugin](#_ibd5r61ikihy)

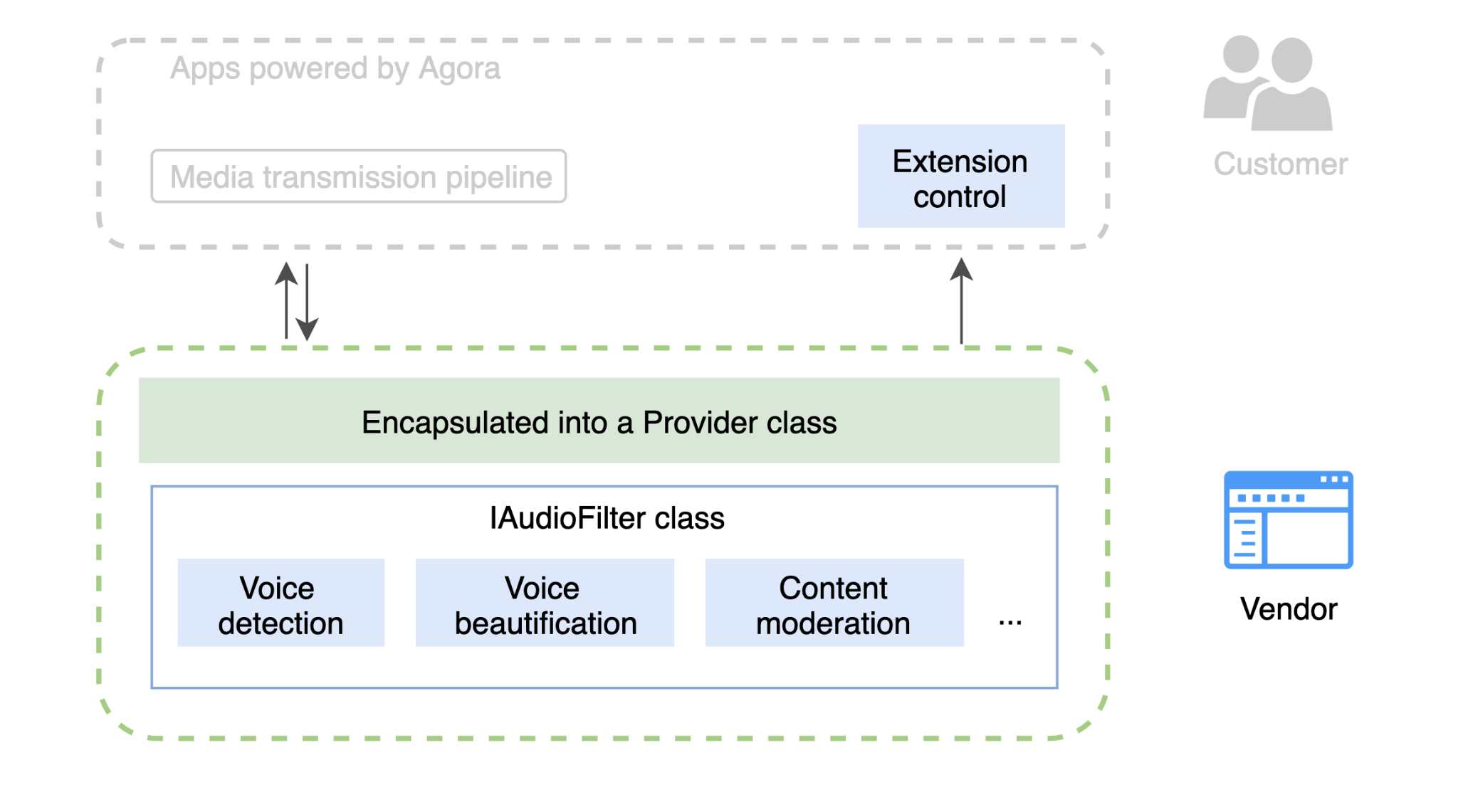
[Reference](#_rkv4v696sro2)

Making an audio-filter plugin enables you to encapsulate your audio-processing capabilities, such as voice beautification and voice detection, into an interface. Customers with real-time engagement apps powered by Agora can use these capabilities for more engaging and diversified use cases by simply calling an API.

This page shows you how to use the APIs provided by Agora Extension Marketplace to encapsulate your audio processing capabilities into an interface.

## Understand the tech

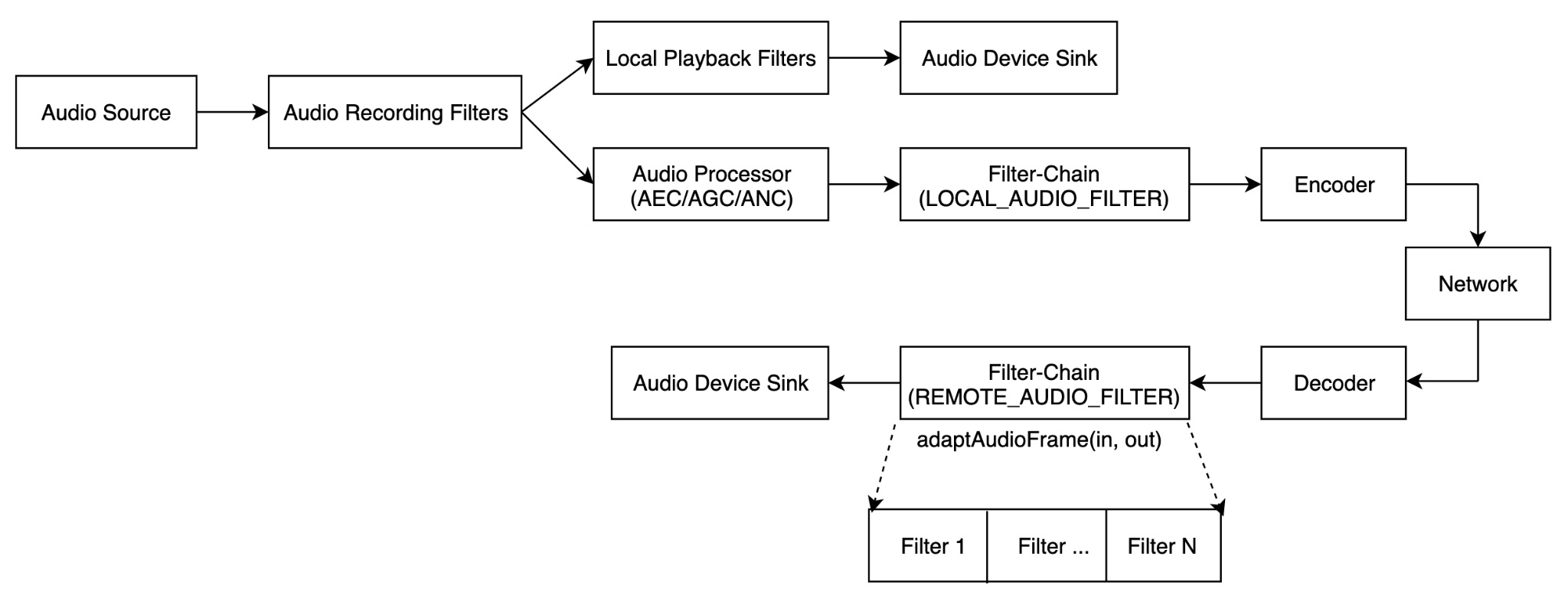
The following diagram shows how Agora enables you to wrap your audio-processing capabilities into a plugin:



Agora provides the following interfaces for building an audio-filter plugin:

* IAudioFilter, which implements receiving audio data, processing them, and delivering them to the transmission pipeline.
* IExtensionProvider, which encapsulates the functions in IAudioFilter into a Provider class, and enables the interaction between the plugin and the app.

The following diagram shows how audio filters work in the media transmission pipeline:



## Prerequisites

Before processing, ensure that your development environment meets the following requirements:

**Android**

* Android Studio 3.0 or later.
* Android SDK API Level 16 or higher.

**iOS**

* Xcode 9.0 or later (the interface description in this article is based on Xcode 11.0).
* An iOS device running iOS 9.0 or later.

## Project setup

Follow the steps to integrate the SDK into your project.

**Android**

1. Download [Agora Native SDK for Android](https://download.agora.io/sdk/release/Agora_Native_SDK_for_Android_arsenal_14052_20210708_0048.zip) and extract the downloaded zip file.
2. Copy the header files in the rtc/sdk/low\_level\_api/include directory and save them under the directory of your project file.

**iOS**

1. Download [Agora Native SDK for iOS](https://download.agora.io/sdk/release/Agora_Native_SDK_for_iOS_arsenal_39288_20210708_0119.zip) and extract the Agora\_Native\_SDK\_for\_iOS zip file
2. Navigate to **TARGETS > Project Name > General > Frameworks, Libraries, and Embedded Content** in your **XCode**, and click **+ > Add Other... > Add Files** to import the libs/AgoraRtcKit.framework file into your project. Ensure that you set the **Embed** option as **Do Not Embed**.

## Build an audio filter plugin

### **Implement IAudioFilter**

You can use the IAudioFilter interface to implement an audio filter plugin. You can find the interface in the NGIAgoraMediaNode.h file. You need to implement this interface first, and you must implement at least the following methods from this class:

* adaptAudioFrame
* setEnabled
* isEnabled
* setProperty
* getProperty
* getName

**adaptAudioFrame**

Adapts the audio frame. This is the core method of the IAudioFilter interface. By calling this method, the SDK processes audio frames from inAudioFrame and returns the adapted frames with adaptedFrame. This method supports audio data in the PCM format only.

| bool adaptAudioFrame(const media::base::AudioPcmFrame& inAudioFrame, media::base::AudioPcmFrame& adaptedFrame) |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| inAudioFrame | An input parameter. The pointer to the audio frames to be processed. |
| adaptedFrame | An output parameter. The pointer to the processed audio frames. |

**setEnabled**

Enables or disables the audio filter.

| virtual void setEnabled(bool enable) {} |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| enable | Whether to enable the audio filter:   * true: Enable the audio filter. * false: (Default) Do not enable the audio filter. |

**isEnabled**

Checks whether the audio filter is enabled.

| virtual bool isEnabled() { return true; } |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| Return value | Whether the audio filter is enabled:   * true: The audio filter is enabled. * false: The audio filter is not enabled. |

**setProperty**

Sets the property of the audio filter plugin.

| size\_t setProperty(const char\* key, const void\* buf, size\_t buf\_size) |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| key | The key of the property. |
| buf | The buffer of the property in the JSON format. You can use the open source nlohmann/json library for the serialization and deserialization between the C++ struct and the JSON string. |
| buf\_size | The size of the buffer. |

**getProperty**

Gets the property of the audio filter plugin.

| size\_t getProperty(const char\* key, char\* property, size\_t buf\_size) |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| key | The key of the property. |
| property | The pointer to the property. |
| buf\_size | The size of the buffer. |

**getName**

Retrieves the vendor name. You need to set the vendor name in the return value of this method.

| virtual const char \* getName() const = 0; |
| --- |

**Sample code**

The following code sample shows how to use these methods together to implement an audio filter plugin.

| // After receiving the audio frames to be processed, call adaptAudioFrame to process the audio frames. bool ExtensionAudioFilter::adaptAudioFrame(const media::base::AudioPcmFrame& inAudioPcmFrame,  media::base::AudioPcmFrame& adaptedPcmFrame) {  return audioProcess\_->processFrame(inAudioPcmFrame, adaptedPcmFrame) == 0; }   // Call setProperty to set the property of the audio filter. int ExtensionAudioFilter::setProperty(const char\* key, const void\* buf, int buf\_size) {  std::string str\_volume = "100";  if (std::string(key) == "volume") {  str\_volume = std::string(static\_cast<const char\*>(buf), buf\_size);  }    int int volume\_ = atoi(str\_volume.c\_str());  audioProcessor\_->setVolume(int\_volume\_);  return ERR\_OK; }   // Call getProperty to get the property of the audio filter. int ExtensionAudioFilter::getProperty(const char\* key, void\* buf, int buf\_size) const override {return ERR\_OK; }   // Call setEnabled to enable the audio filter. void ExtensionAudioFilter::setEnabled(bool enable) override { enabled\_ = enable; } // Call isEnabled to check whether the audio filter is enabled. bool ExtensionAudioFilter::isEnabled() const override {return enabled\_; }   // Set the vendor name in the return value of getName. const char\* getName() const override { return filterName\_.c\_str(); } |
| --- |

### **Encapsulate the audio filter plugin**

To encapsulate the audio filter plugin, you need to implement the IExtensionProvider interface. You can find the interface in the NGIAgoraExtensionProvider.h file. The following methods from this interface must be implemented:

* enumerateExtensions
* setExtensionControl
* createAudioFilter

**enumerateExtensions**

Enumerates your plugins that can be encapsulated. The SDK triggers this callback when loading the plugin. In the callback, you need to return information about all of your plugins that can be encapsulated.

| virtual void enumerateExtensions(  ExtensionMetaInfo\* extension\_list, int& extension\_count) {  (void) extension\_list;  extension\_count = 0;  } |
| --- |

| **参数** | **描述** |
| --- | --- |
| extension\_list | Plugin information, including plugin type and name. For details, see the definition of ExtensionMetaInfo. |
| extension\_count | The total number of the plugins that can be encapsulated. |

The definition of ExtensionExtensionMetaInfo is as follows:

| // Plugin type represents where the plugin is located in the media transmission pipeline enum EXTENSION\_TYPE {  // Audio processing filter  AUDIO\_FILTER,  // Video preprocessing filter  VIDEO\_PRE\_PROCESSING\_FILTER,  // Video postprocessing filter  VIDEO\_POST\_PROCESSING\_FILTER,  // Reserved for future use  AUDIO\_SINK,  // Reserved for future use  VIDEO\_SINK,  // Reserved for future use  UNKNOWN,  };     // Plugin information, including plugin type and name struct ExtensionMetaInfo {  EXTENSION\_TYPE type;  const char\* extension\_name;  }; |
| --- |

If you specify the plugin type as LOCAL\_AUDIO\_FILTER as the extension type, after the customer creates the IExtensionProvider object when initializing RtcEngine, the SDK calls the createAudioFilter method, and you need to return the IAudioFilter instance in this method.

**setExtensionControl**

Sets the extension control.

| virtual void setExtensionControl(IExtensionControl\* control) |
| --- |

After calling this method, you need to maintain the IExtensionControl object returned in this method. The IExtensionControl object manages the interaction between the plugin and the app by triggering callbacks and sending SDK logs. For example, if you have called fireEvent in IExtensionControl:

| void ByteDanceProcessor::dataCallback(const char\* data){  if (control\_ != nullptr) {  control\_->fireEvent(id\_, "beauty", data);   } } |
| --- |

And if the app registers the IMediaExtensionObserver class when initializing RtcEngine, the SDK triggers the following callback on the app client:

| @Override public void onEvent(String vendor, String key, String value) { ...... } |
| --- |

**createAudioFilter**

Creates an audio filter. You need to pass the IAudioFilter instance in this method.

| virtual agora\_refptr<IAudioFilter> createAudioFilter() |
| --- |

After creating an audio filter object, the plugin processes the input audio frames with methods in IAudioFilter.

**Sample code**

The following code sample shows how to use these methods to encapsulate the audio filter:

| void ExtensionProvider::enumerateExtensions(ExtensionMetaInfo\* extension\_list,  int& extension\_count) {  extension\_count = 2;  //Declare a Video Filter, and IExtensionProvider::createVideoFilter will be called  ExtensionMetaInfo i;  i.type = EXTENSION\_TYPE::VIDEO\_PRE\_PROCESSING\_FILTER;  i.extension\_name = agora::extension::VIDEO\_FILTER\_NAME;  extension\_list[0] = i;   //Declare an Audio Filter, and IExtensionProvider::createAudioFilter will be called  ExtensionMetaInfo j;  j.type = EXTENSION\_TYPE::AUDIO\_FILTER;  j.extension\_name = agora::extension::AUDIO\_FILTER\_NAME;  extension\_list[1] = j; }  agora\_refptr<agora::rtc::IAudioFilter> ExtensionProvider::createAudioFilter(const char\* name) {  PRINTF\_INFO("ExtensionProvider::createAudioFilter %s", name);  auto audioFilter = new agora::RefCountedObject<agora::extension::ExtensionAudioFilter>(name, audioProcessor\_);  return audioFilter; }  void ExtensionAudioProvider::setExtensionControl(rtc::IExtensionControl\* control){  audioProcessor\_->setExtensionControl(control); } |
| --- |

### **Package the plugin**

**Android**

After encapsulating the audio filter plugin, you need to register and package it into a .aar or .so file, and submit it together with a file that contains the extension name, vendor name and filter name to Agora.

1. Register the plugin

Register the plugin with the macro REGISTER\_AGORA\_EXTENSION\_PROVIDER, which is in the AgoraExtensionProviderEntry.h file.

Use this macro at the entrance of the plugin implementation. When the SDK loads the plugin, this macro automatically registers it to the SDK. Code sample:

| REGISTER\_AGORA\_EXTENSION\_PROVIDER(ByteDance, agora::extension::ExtensionProvider); |
| --- |

2. Link the .so file

In CMakeLists.txt, specify the save path for the libagora-rtc-sdk-jni.so file in the downloaded SDK package according to the following table:

| **File** | **Path** |
| --- | --- |
| 64-bit libagora-rtc-sdk-jni.so | AgoraWithByteDanceAndroid/agora-bytedance/src/main/agoraLibs/arm64-v8a |
| 32-bit libagora-rtc-sdk-jni.so | AgoraWithByteDanceAndroid/agora-bytedance/src/main/agoraLibs/arm64-v7a |

3. Provide extension information

Create a .java or .md file and fill the following information:

* Extension\_NAME: The name of the target link library used in CMakeLists.txt. For example, for a .so file named libagora-bytedance.so, the EXTENSION\_NAME should be agora-bytedance.
* EXTENSION\_VENDOR\_NAME: The name of the extension provider, which is used for registering in the agora-bytedance.cpp file.
* EXTENSION\_FILTER\_NAME: The name of the filter, which is defined in ExtensionProvider.h.

**iOS**

After encapsulating the audio filter plugin, you need to register and package it into a .xcframework or .framework file, and submit it to Agora.

Register the plugin with the macro REGISTER\_AGORA\_EXTENSION\_PROVIDER, which is in the AgoraExtensionProviderEntry.h file.

Use this macro at the entrance of the plugin implementation. When the SDK loads the plugin, this macro automatically registers it to the SDK. Code sample:

| REGISTER\_AGORA\_EXTENSION\_PROVIDER(ByteDance, agora::rtc::BDVideoFilterProviderCpp); |
| --- |

## Reference

Agora also provides demo projects that implement and encapsulate the audio plugin. You can download the sample project or view the source code as a reference:

* Android: <https://download.agora.io/demo/release/AgoraWithByteDanceAndroid.zip>
* iOS: <https://download.agora.io/demo/release/AgoraWithByteDanceiOS.zip>