# Video Filter Plugins

[Video Filter Plugins](#_ieohd86gy4t7)

[Understand the tech](#_v2juc4z4fn76)

[Prerequisites](#_ipmv9tinpciw)

[Project setup](#_av0apkwuycb)

[Build a video filter plugin](#_4bl5ou3urptb)

[Implement the plugin](#_mbgb2actzdi1)

[getProcessMode](#_b83oaygqjznv)

[start](#_f2b79dpm8ci)

[stop](#_5b10ptt6avt)

[getVideoFormatWanted](#_op6zw7h9mmtw)

[pendVideoFrame](#_cn3r2xx6plp9)

[setProperty](#_vqa2f0h2c51o)

[Sample code](#_6zou8bukzy97)

[Encapsulate the video filter plugin](#_inlvbicywtyz)

[enumerateExtensions](#_x52a9ixx89ua)

[createVideoFilter](#_himhfmupc4xc)

[Sample code](#_wgrw1b2occk4)

[Package the plugin](#_5h4vi0vwvkxe)

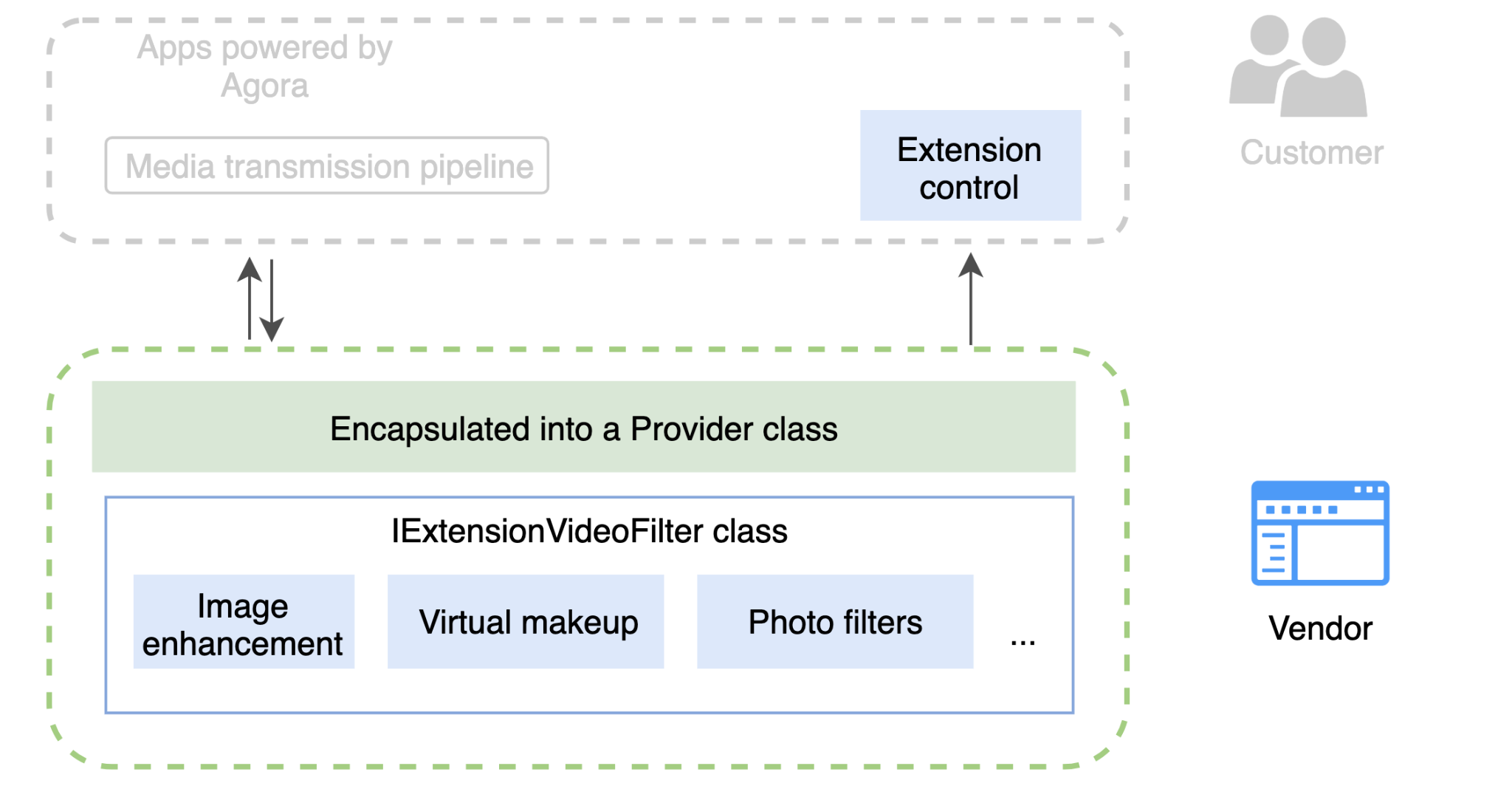
[Reference](#_vaqs2cprljwe)

Making a video-filter plugin enables you to encapsulate your video-processing capabilities, such as image enhancement and virtual makeup, 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 video processing capabilities into an interface.

## Understand the tech

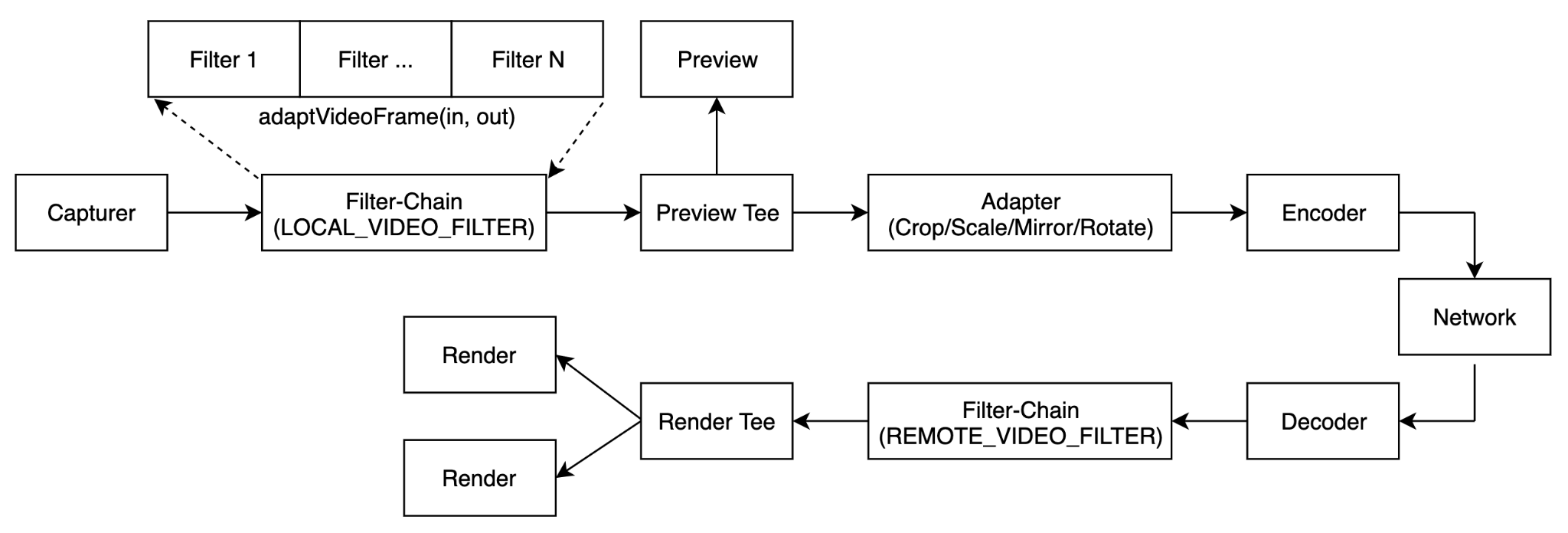
The following diagram shows how Agora enables you to wrap your video-processing capabilities into a plugin:



Agora provides the following interfaces for building a video-filter plugin:

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

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



## Prerequisites

Before proceeding, ensure that you have the following:

**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 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 zip file
2. In **Xcode**, navigate to **TARGETS > Project Name > General > Frameworks, Libraries, and Embedded Content**, 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 a video filter plugin

### **Implement the plugin**

To implement your video filter plugin, you need to implement the IExtensionVideoFilter class. You can find the class in the NGIAgoraMediaNode.h file. IExtensionVideoFilter contains the following methods:

* [getProcessMode](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-getprocessmode)
* [start](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-start)
* [stop](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-sto)
* [getVideoFormatWanted](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-getvideoformatwanted)
* [adaptVideoFrame](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-adaptvideoframe)
* [pendVideoFrame](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-pendvideof)
* [deliverVideoFrame](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-delivervideofra)
* [setProperty](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-setproperty)
* [getProperty](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-getproperty)

#### **getProcessMode**

Sets how the SDK communicates with your video filter plugin. The SDK triggers this callback first when loading the plugin. After receiving the callback, you need to return mode and independent\_thread to specify how the SDK communicates with the plugin.

| virtual void getProcessMode(ProcessMode& mode, bool& independent\_thread) = 0; |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| mode | The mode for transferring video frames between the SDK and plugin. You can set it to the following values:   * Sync: Synchronous mode, where the SDK and plugin transfer video frames through adaptVideoFrame. * Async: Asynchronous mode, where the SDK sends video frames to the plugin through pendVideoFrame, and the plugin returns processed video frames to the SDK through deliverVideoFrame. |
| independent\_thread | Whether to create an independent thread for the plugin:   * true: Create an independent thread for the plugin, so that the SDK sends all the callbacks to the plugin in the created thread. * false: Do not create an independent thread for the plugin. In this case, the SDK sends all the callbacks to the plugin in its original video processing thread. |

You can set the value of mode and independent\_thread based on the complexity of your YUV algorithm and whether OpenGL is used:

* + If your plugin uses complicated YUV algorithm, Agora recommends setting mode to Async and independent\_thread to false; if your plugin does not use complicated YUV algorithm, Agora recommends setting mode to Sync and independent\_thread to false.
  + If your plugin uses OpenGL for data processing, Agora recommends setting mode to Sync and independent\_thread to true.

#### **start**

| virtual int start(agora::agora\_refptr<Control> control) = 0; |
| --- |

The SDK triggers this callback after the video transmission pipeline starts. You can initialize OpenGL in this callback.

The SDK also passes a Control object to the plugin in this method. The Control class provides methods for the plugin to interact with the SDK. You can implement the methods in the Control class based on your actual needs:

| class Control : public RefCountInterface {  public:  /\*\*  \* In asynchronous mode (mode is set to Async), the plugin calls this methods to return the processed video frame to the SDK.  \* Before calling this method, ensure that the SDK submits the video frame to the plugin through pendVideoFrame.  \*/  virtual ProcessResult deliverVideoFrame(agora::agora\_refptr<IVideoFrame> frame) = 0;  /\*\*  \* If the plugin needs a new memory pool, call this method to create a new IVideoFrame object for better memory management.  \* For example, an image enhancement plugin can call this method to save both the original frame and processed frame with more efficient memory management.  \*/  virtual agora::agora\_refptr<IVideoFrameMemoryPool> getMemoryPool() = 0;  /\*\*  \* Call this method to report an event to the SDK. The SDK then sends the event notification to the app.  \*/  virtual int postEvent(const char\* key, const char\* value) = 0;  /\*\*   \* Call this method to print logs to the SDK.  \*/  virtual void printLog(commons::LOG\_LEVEL level, const char\* format, ...) = 0;  /\*\*  \* If an unrecoverable error occurs within the plugin, call this method to report the error and  \* stop SDK from sending video frames to the plugin. The SDK then passes the error message to the app.   \*/  virtual void disableMe(int error, const char\* msg) = 0;  }; |
| --- |

#### **stop**

| virtual int stop() = 0; |
| --- |

The SDK triggers this callback before the video transmission pipeline stops, . You can release OpenGL in this callback.

#### **getVideoFormatWanted**

Sets the type and format of the video frame sent to your plugin. The SDK triggers this callback before sending a video frame to the plugin. In the callback, you need to specify the type and format for the frame. You can change the type and format of subsequent frames when you receive the next callback.

| virtual void getVideoFormatWanted(VideoFrameData::Type& type, RawPixelBuffer::Format& format) = 0; |
| --- |

| **Parameter** | **Description** |
| --- | --- |
| type | The type of the video frame. Currently you can only set it to RawPixels, which means raw data. |
| format | The format of the video frame. You can set it to the following values:   * Unknown：An unknown format. * I420：The I420 format. * I422：The I420 format. * NV21：The NV21 format. * NV12：The NV12 format. * RGBA：The RGBA format. * ARGB：The AGRB format. * BGRA：The BGRA format. |

**adaptVideoFrame**

Adapts the video frame. In synchronous mode (mode is set to Sync), the SDK and plugin transfer video frames through this method. By calling this method, the SDK sends video frames to the plugin with in, and the plugin returns the processed frames with out.

| virtual ProcessResult adaptVideoFrame(agora::agora\_refptr<IVideoFrame> in, agora::agora\_refptr<IVideoFrame>& out) {  return ProcessResult::kBypass;  } |
| --- |

**Parameters**

| **Parameter** | **Description** |
| --- | --- |
| in | An input parameter. The video frame to be processed by the plugin. |
| out | An output parameter. The processed video frame. |

**Returns**

The result of processing the video frame:

* Success: The plugin has processed the frame successfully.
* ByPass: The plugin does not process the frame and passes it to the subsequent link in the filter chain.
* Drop: The plugin discards the frame.

#### **pendVideoFrame**

Submits the video frame. In asynchronous mode (mode is set to Async), the SDK submits the video frame to the plugin through this method. After calling this method, the plugin must return the processed video frame through deliverVideoFrame in the Control class.

| virtual ProcessResult pendVideoFrame(agora::agora\_refptr<IVideoFrame> frame) {  return ProcessResult::kBypass;  } |
| --- |

**Parameter**

| **Parameter** | **Description** |
| --- | --- |
| frame | The video frame to be processed by the plugin. |

**Returns**

The result of processing the video frame:

* Success: The plugin has processed the frame successfully.
* ByPass: The plugin does not process the frame and passes it to the subsequent link in the chain.
* Drop: The plugin discards the frame.

#### **setProperty**

Sets the property of the video filter plugin. When an app client calls setProperty, the SDK triggers this callback. In the callback, you need to return the plugin property.

| int ExtensionVideoFilter::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 video filter plugin. When the app client calls getProperty, the SDK calls this method to get the plugin property.

| int ExtensionVideoFilter::getProperty(const char \*key, void \*buf, 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. |

#### 

#### **Sample code**

The following code sample shows how to use these APIs together to implement a video filter plugin:

| #include "ExtensionVideoFilter.h" #include "../logutils.h" #include <sstream>   namespace agora {  namespace extension {    ExtensionVideoFilter::ExtensionVideoFilter(agora\_refptr<ByteDanceProcessor> byteDanceProcessor):threadPool\_(1) {  byteDanceProcessor\_ = byteDanceProcessor;  }    ExtensionVideoFilter::~ExtensionVideoFilter() {  byteDanceProcessor\_->releaseOpenGL();  }      // Sets how the SDK communicates with your video filter plugin.  void ExtensionVideoFilter::getProcessMode(ProcessMode& mode, bool& independent\_thread) {  mode = ProcessMode::kSync;  independent\_thread = false;  mode\_ = mode;  }      // Sets the type and format of the video frame sent to your plugin.  void ExtensionVideoFilter::getVideoFormatWanted(rtc::VideoFrameData::Type& type,  rtc::RawPixelBuffer::Format& format) {  type = rtc::VideoFrameData::Type::kRawPixels;  format = rtc::RawPixelBuffer::Format::kI420;  }    // Save the Control object and initialize OpenGL  int ExtensionVideoFilter::start(agora::agora\_refptr<Control> control) {  PRINTF\_INFO("ExtensionVideoFilter::start");  if (!byteDanceProcessor\_) {  return -1;  }  if (control) {  control\_ = control;  byteDanceProcessor\_->setExtensionControl(control);  }  if (mode\_ == ProcessMode::kAsync){  invoker\_id = threadPool\_.RegisterInvoker("thread\_videofilter");  auto res = threadPool\_.PostTaskWithRes(invoker\_id, [byteDanceProcessor=byteDanceProcessor\_] {  return byteDanceProcessor->initOpenGL();  });  isInitOpenGL = res.get();  } else {  isInitOpenGL = byteDanceProcessor\_->initOpenGL();  }  return 0;  }    // Release OpenGL  int ExtensionVideoFilter::stop() {  PRINTF\_INFO("ExtensionVideoFilter::stop");  if (byteDanceProcessor\_) {  byteDanceProcessor\_->releaseOpenGL();  isInitOpenGL = false;  }  return 0;  }      // When mode is set to Async, the SDK and plugin transfer video frames through pendVideoFrame and deliverVideoFrame.  rtc::IExtensionVideoFilter::ProcessResult ExtensionVideoFilter::pendVideoFrame(agora::agora\_refptr<rtc::IVideoFrame> frame) {  if (!frame || !isInitOpenGL) {  return kBypass;  }    bool isAsyncMode = (mode\_ == ProcessMode::kAsync);  if (isAsyncMode && byteDanceProcessor\_ && control\_ && invoker\_id >= 0) {  threadPool\_.PostTask(invoker\_id, [videoFrame=frame, byteDanceProcessor=byteDanceProcessor\_, control=control\_] {  rtc::VideoFrameData srcData;  videoFrame->getVideoFrameData(srcData);  byteDanceProcessor->processFrame(srcData);  control->deliverVideoFrame(videoFrame);  });  return kSuccess;  }  return kBypass;  }    // When mode is set to Sync, the SDK and plugin transfer video frames through adaptVideoFrame.  rtc::IExtensionVideoFilter::ProcessResult ExtensionVideoFilter::adaptVideoFrame(agora::agora\_refptr<rtc::IVideoFrame> src,  agora::agora\_refptr<rtc::IVideoFrame>& dst) {  if (!isInitOpenGL) {  return kBypass;  }  bool isSyncMode = (mode\_ == ProcessMode::kSync);  if (isSyncMode && byteDanceProcessor\_) {  rtc::VideoFrameData srcData;  src->getVideoFrameData(srcData);  byteDanceProcessor\_->processFrame(srcData);  dst = src;  return kSuccess;  }  return kBypass;  }      // Set the property of the video filter.  int ExtensionVideoFilter::setProperty(const char \*key, const void \*buf,  size\_t buf\_size) {  PRINTF\_INFO("setProperty %s %s", key, buf);  std::string stringParameter((char\*)buf);  byteDanceProcessor\_->setParameters(stringParameter);  return 0;  }    // Get the property of the video filter.  int ExtensionVideoFilter::getProperty(const char \*key, void \*buf, size\_t buf\_size) {  return -1;  }  } } |
| --- |

**Encapsulate the plugin**

To encapsulate the video 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](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-enumerateextensions)
* [createVideoFilter](https://confluence.agoralab.co/pages/editpage.action?pageId=731595171#VideoFilterPlugins-createvideofilt)

#### **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. |
| extension\_count | The total number of the plugins that can be encapsulated. |

Plugin information is defined 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 return the plugin type as VIDEO\_PRE\_PROCESSING\_FILTER or VIDEO\_POST\_PROCESSING\_FILTER, the SDK calls the createVideoFilter method after the customer creates the IExtensionVideoProvider object when initializing RtcEngine.

#### **createVideoFilter**

Creates a video filter. When the SDK calls this method, you need to return the IExtensionVideoFilter instance.

| virtual agora\_refptr<IExtensionVideoFilter> createVideoFilter(const char\* name) {  return NULL;  } |
| --- |

After the IExtensionVideoFilter instance is created, the plugin processes video frames with methods in the IExtensionVideoFilter class.

#### **Sample code**

The following code sample shows how to use these APIs to encapsulate the video filter:

| #include "ExtensionProvider.h" #include "../logutils.h" #include "VideoProcessor.h" #include "plugin\_source\_code/JniHelper.h"  namespace agora {  namespace extension {  ExtensionProvider::ExtensionProvider() {  PRINTF\_INFO("ExtensionProvider create");  byteDanceProcessor\_ = new agora::RefCountedObject<ByteDanceProcessor>();  audioProcessor\_ = new agora::RefCountedObject<AdjustVolumeAudioProcessor>();  }   ExtensionProvider::~ExtensionProvider() {  PRINTF\_INFO("ExtensionProvider destroy");  byteDanceProcessor\_.reset();  audioProcessor\_.reset();  }    // Enumerate your plugins that can be encapsulated  void ExtensionProvider::enumerateExtensions(ExtensionMetaInfo\* extension\_list,  int& extension\_count) {  extension\_count = 1;  ExtensionMetaInfo i;  i.type = EXTENSION\_TYPE::VIDEO\_PRE\_PROCESSING\_FILTER;  i.extension\_name = agora::extension::VIDEO\_FILTER\_NAME;  extension\_list[0] = i;  }   // Create a video filter plugin  agora\_refptr<agora::rtc::IExtensionVideoFilter> ExtensionProvider::createVideoFilter(const char\* name) {  PRINTF\_INFO("ExtensionProvider::createVideoFilter %s", name);  auto videoFilter = new agora::RefCountedObject<agora::extension::ExtensionVideoFilter>(byteDanceProcessor\_);  return videoFilter;  }   void ExtensionProvider::setExtensionControl(rtc::IExtensionControl\* control){  }  } } |
| --- |

### **Package the plugin**

**Android**

After encapsulating the video 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](http://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](http://libagora-rtc-sdk-jni.so/) | AgoraWithByteDanceAndroid/agora-bytedance/src/main/agoraLibs/arm64-v8a |
| 32-bit [libagora-rtc-sdk-jni.so](http://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](http://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 video 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 video 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>