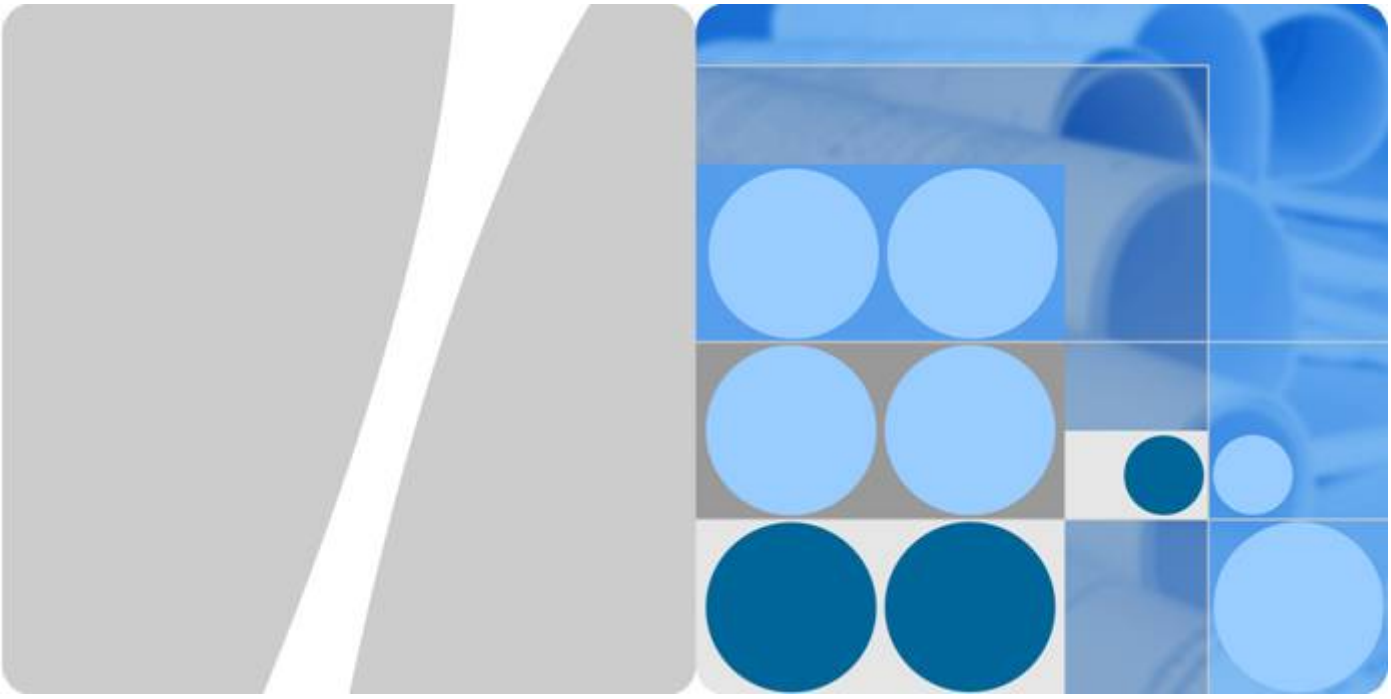


Huawei HiLens

# SDK Reference

Issue 01

Date 2023-05-30

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

### Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

### Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

# Contents



# 1 Before You Start

This document describes how to install and configure a development environment and invoke functions provided through HiLens Framework SDKs for secondary development.

**Table 1-1** Documentation guide

| Chapter | Description |
| --- | --- |
| **HiLens Framework_C++** | HiLens Framework development tool package |
| **Environment Preparations** | How to prepare for using the HiLens Framework development tool packages |
| **Video Input**<br>**Audio Input**<br>**Pre-processing**<br>**Model Management**<br>**Output**<br>**Resource Management**<br>**Log** | Classes and functions encapsulated in the HiLens Framework |
| **Error Codes** | Error code description and recommended solutions |

# 2 HiLens Framework_C++

The HiLens Framework Software Development Kit (SDK) is a C++ development package running on HiLens Kit cameras. Users can use the HiLens SDKs to develop C++ skills and run them on HiLens Kit cameras.

## HiLens Framework Introduction

The HiLens Framework encapsulates bottom-layer APIs to implement common management functions, enabling developers to easily develop skills and cultivate the AI ecosystems on the HiLens management console.

**Figure 2-1** shows the layered structure of HiLens Framework. HiLens Framework encapsulates the underlying multimedia processing libraries (cameras/microphone driver module Media_mini), the image processing library (DVPP) related to D chips, and the model management library (ModelManager). Developers can also use the familiar visual processing library OpenCV. HiLens Framework provides the modules in **Table 2-1** for developers, such as human figure detection and fatigue driving detection.

**Figure 2-1** HiLens Framework

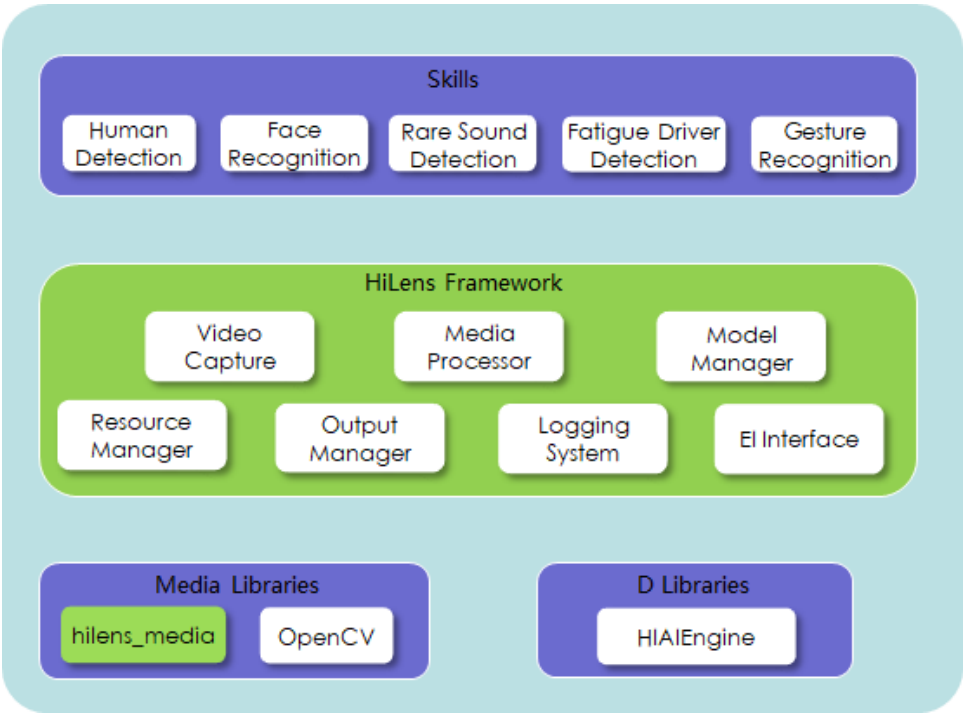



**Table 2-1** Module description

| No. | Module | Function |
| --- | --- | --- |
| 1 | Input Manager | Manages the access of input data such as video and audio data. |
| 2 | Media Processor | Processes media data such as video and audio data. |
| 3 | Model Manager | Initializes models and runs inference tasks. |
| 4 | Output Manager | Manages output tasks such as streams, files, and message notifications. |
| 5 | Resource Manager | Manages paths of resources such as files, images, and models. |
| 6 | Logging System | Manages the log system. |

# 3 Preparations

## 3.1 Environment Preparations

### Deploying an Environment

1. Check your compiler's Linux system environment.

   You can run the **uname -a** command to view the Linux system information. For example, to view the Ubuntu system information of x86_64, run the following command:

   **Linux ubuntu 4.4.0-144-generic #170-Ubuntu SMP Thu Mar 14 11:56:20 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux**

2. Check the system environment of the HiLens Kit device.

   For details on how to log in to the operating system of the device, see the "Connecting to a HiLens Kit Device Using SSH" section in *HiLens User Guide*.

   For example, run the **uname -a** command to obtain the following HiLens Kit system environment:

   **Linux Euler 4.19.36-vhulk1907.1.0.h448.eulerosv2r8.aarch #1 SMP Mon Jul 22 00.00.00 UTC 2019 aarch64 aarch64 aarch64 GNU/Linux**

3. Download and decompress the cross compilation tool.

   If your compiler runs x86_64 Linux, you can directly download the **cross compilation tool**, which contains the libraries needed by HiLens Kit devices.

4. Go to the directory of the cross compilation tool, run the **pwd** command to obtain the path, and set the compiler path during compilation.

   If your compiler runs the x86_64 Linux OS and the path is **.../aarch64-linux-gnu-gcc-7.3.0**, the path is set as follows:

   **export CC=".../aarch64-linux-gnu-gcc-7.3.0/bin/aarch64-linux-gnu-gcc"**

   **export CXX=".../aarch64-linux-gnu-gcc-7.3.0/bin/aarch64-linux-gnu-g++"**

   For details about the compilation guide, see the sample code in step **6**.

5. Download the HiLens Framework SDK development package **cloud-c-sdk-HiLensFramework-1.0.4.tar.gz**. Decompress and rename the file.

   For details about how to download it, see **SDK Downloading**.

6. Develop code by referring to the sample and API description. For the compilation guide and sample code description, see **README.md** in the sample code.

   **HiLens Framework Sample Code**

   NOTE

   When setting up the environment, you must install and debug the SDK development package and reference sample program on the compiler.

# 3.2 SDK Downloading

Download the SDK software package based on the HiLens Framework firmware version. See **Table 3-1**. You can log in to the HiLens console, choose **Device Management** > **HiLens Kit**, and **view the HiLens Framework firmware version**.

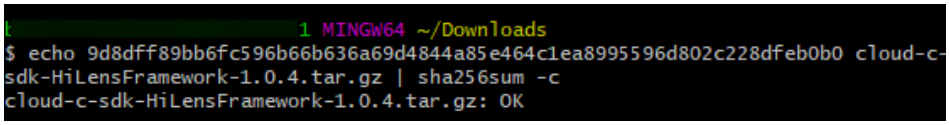
**Table 3-1** Downloading SDKs

| Firmware Version | SDK Download Link |
| --- | --- |
| 1.0.6 or earlier | **HiLens Framework SDK** (sha256:9d8dff89bb6fc596b66b636a69d4844a85e464c1ea8995596d802c228dfeb0b0) |
| 1.0.7 | **HiLens Framework SDK** (sha256:690535e4682b8f008cf48cfb3ea698950316723a2bd8d2bdd20c9b92d66ed36e) |
| Version 1.0.8 | **HiLens Framework SDK** (sha256:36a4476fe5dc0766c5a3e4d53792f54be55a69548eb43a9d63fa6cb5a312e1fb) |
| 1.1.0 or later | **HiLens Framework SDK** (sha256:57278a0d10d37031dd158094a3930dc9ba867b9d0362c833c9bf830a258fc94f) |

After the SDK is downloaded, verify the signature in the directory where the file is stored (Git is recommended for Windows). If the verification is successful, **OK** is returned. Run the following command:

```
echo [sha256 value] [firmware package] | sha256sum -c
```

**Figure 3-1** Signature verification example



# 4 Initialization

## 4.1 Initializing the HiLens Framework

This section describes how to initialize the HiLens Framework. Before calling other APIs of HiLens Framework, you need to perform global initialization.

### API Calling

HiLensEC hilens::Init(const std::string & verify)

### Parameter Description

**Table 4-1** Parameters

| Parameter | Description |
| --- | --- |
| verify | The value must be the same as that of **Verification Value** you entered in the **Basic Information** area when you create a skill on the HiLens management console. If they are different, HiLens Framework forcibly stops the skill. |

### Return Values

If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

## 4.2 Releasing the HiLens Framework

This section describes how to call the following API to release related resources.

## API Calling

HiLensEC hilens::Terminate()

## Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

# 5 Video Input

## 5.1 Introduction

### HiLens::VideoCapture Class

This section describes how to use a video collector to read data from a local camera or an IP camera.

```
#include <video_capture.h>
```

### Constructor and Destructor

```
~VideoCapture()
virtual hilens::VideoCapture::~VideoCapture()
```

## 5.2 Constructing Video Collectors

### Local Camera

This part describes how to construct a video collector (local camera). If the operation fails, you can view the skill logs or output to locate the error cause.

- **API calling**

  static std::shared_ptr<VideoCapture> hilens::VideoCapture::Create()

- **Return value**

  If the operation is successful, a video collector instance is returned. If the operation fails, **nullptr** is returned.

### IP Camera

This part describes how to construct a video collector (IP camera). If the operation fails, you can view the skill logs or output to locate the error cause.

- **API calling**

  static std::shared_ptr<VideoCapture> hilens::VideoCapture::Create(const std::string & name)

static std::shared_ptr<VideoCapture> hilens::VideoCapture::Create(const std::string & name, const unsigned int width, const unsigned int height)

- **Parameter description**

**Table 5-1** Parameters

| Parameter | Description |
| --- | --- |
| name | Camera name in the device configuration file (IPC in the device configurations). The camera name in the device configuration file is preferentially used. You can also enter a stream obtaining address in the format of **rtsp://***xxx*. In 1.0.7 and later versions, the local MP4 files can be directly read, and the width and height of the read video frame can be set.<br>You can configure the camera name on the HiLens management console. For details, see the "Configuring Cameras" section in *HiLens User Guide*. |
| width | Width of the read video frame. The width must be a multiple of 16. The recommended value is a multiple of 32, and the minimum value is 128. This parameter is supported only by 1.0.7 and later versions. |
| height | Height of the read video frame. The value must be a multiple of 2 and the minimum value is 128. This parameter is supported only by 1.0.7 and later versions. |

## USB Camera

This part describes how to construct a video collector (USB camera). If the operation fails, the "CreateError" message is displayed, and you can view the skill logs or output to locate the error cause. Currently, only one UVC camera can be inserted. The camera ID is **0**.

- **API calling**

  static std::shared_ptr hilens::VideoCapture::Create(int dev)

- **Parameter description**

**Table 5-2** Parameters

| Parameter | Description |
| --- | --- |
| dev | ID of the UVC camera in **/dev** of the HiLens Kit system.<br>For details on how to log in to the HiLens Kit system, see the "Connecting to a HiLens Kit Device Using SSH" section in *HiLens User Guide*. |

- **Return value**

  If the operation is successful, a video collector instance is returned. If the operation fails, **nullptr** is returned.

# 5.3 Reading Video Frames from Cameras

This section describes how to read video frames. If an error occurs during camera reading, the exception message **std::runtime_error** will be displayed.

## API Calling

virtual cv::Mat hilens::VideoCapture::Read()

## Return Value

If the camera is an IPC or a local camera, YUV_NV21 data is returned. If the camera is a UVC camera, BGR data is returned.

# 5.4 Obtaining the Video Width

This section describes how to return the width of a video.

## API Calling

virtual int hilens::VideoCapture::Width()

## Return Value

Video width

# 5.5 Obtaining the Video Height

This section describes how to return the height of a video.

## API Calling

virtual int hilens::VideoCapture::Height()

## Return Value

Video height

# 6 Audio Input

## 6.1 Introduction

### hilens::AudioCapture Class

This section describes how to use an audio collector to read data from a local audio file. The related header file has been integrated into **hilens.h**.

```
#include <hilens.h>
```

### Constructor and Destructor

```
~AudioCapture()
virtual hilens::AudioCapture::~AudioCapture()
```

## 6.2 Constructing Audio Collectors

### Local Audio File

Construct an audio collector. If the audio collector fails to be created, you can view skill logs or output to locate the fault. The local microphone uses the default parameters to collect data. The sampling rate is 44100, bit width is 16 bits, dual-channel collection is used, and the number of sampling points in each frame is 1024.

- **API calling**

  1.0.8 or later

  static std::shared_ptr<AudioCapture> hilens::AudioCapture::Create(const std::string filePath)

  1.1.0 or later

  static std::shared_ptr<AudioCapture> Create(const struct AudioProperties& property)

- **Parameter description**

**Table 6-1** Parameters

| Parameter | Description |
| --- | --- |
| filePath | If the parameter is the absolute path of an audio file on the HiLens Kit device (Chinese characters are not supported), audio data is obtained from the file. |
| property | Local microphone recording parameter. The structure is defined as follows:<br>`struct AudioProperties{`<br>`unsigned int enSamplerate;`<br>`unsigned int enBitwidth;`<br>`unsigned int u32PtNumPerFrm;`<br>`unsigned int soundMode;`<br>`}`<br>The value range of each element is as follows:<br>• **enSamplerate**: sampling rate. The value can be **8000**, **12000**, **11025**, **16000**, **22050**, **24000**, **32000**, **44100**, **48000**, **64000**, or **96000**.<br>• **enBitwidth**: bit width. The value is **1**, indicating that the bit width is 16 bits.<br>• **u32PtNumPerFrm**: number of sampling points in each frame. The value range is [80, 2048].<br>• **soundMode**: audio channel mode. The value can be **0** (mono-channel) or **1** (dual-channel). |

NOTE

- The values of **u32PtNumPerFrm** and **enSamplerate** determine the frequency of hardware interrupts. If the frequency is too high, the system performance is affected and other services are affected. It is recommended that the values of **u32PtNumPerFrm** and **enSamplerate** meet the following formula: **(u32PtNumPerFrm x 1,000)/enSamplerate >= 10**. For example, when **enSamplerate** is **16000**, it is recommended that **u32PtNumPerFrm** be greater than or equal to **160**.
- There is only one local microphone. Different recording parameters cannot be set for multiple processes. The parameters set earlier take effect. If the parameter settings are different, the setting fails.
- This API and the API in **Playing an Audio File** cannot be called at the same time.

- **Return Value**

  If the operation is successful, an audio collector instance is returned. If the operation fails, nullptr is returned.

# 6.3 Reading Audio Data

This section describes how to read one or more frames of audio. Only 1.0.8 and later firmware versions are supported.

- **API calling**

  virtual int hilens::AudioCapture::Read(AudioFrame &frames, int n=1)

- **Parameter description**

  The **AudioFrame** structure is defined as follows. For details about the parameters, see **Table 6-2**.

```
typedef struct AudioFrame_s{
  std::shared_ptr<void> data;
  unsigned int size;
}AudioFrame;
```

**Table 6-2** Parameters

| Parameter | Description |
| --- | --- |
| data | Output parameter, indicating the smart pointer for storing the audio data that is read. |
| size | Output parameter, indicating the size of the audio data that is read. |
| n | Input parameter, indicating the number of audio frames read at a time. The maximum value is **512**. |

- **Return Value**

  If the operation is successful, **0** is returned. If the operation fails, **-1** is returned. You can view the log to locate the fault.

# 7 Pre-processing

## 7.1 Introduction

### HiLens::Preprocessor Class

Hardware acceleration preprocessor

```
#include <media_process.h>
```

### Destructor

```
~Preprocessor()
virtual hilens::Preprocessor::~Preprocessor()
```

## 7.2 Constructing Image Preprocessors

This section describes how to construct and initialize a preprocessor for resize/crop operations (3559 hardware acceleration). If the operation fails, you can view the skill log or output to locate the fault.

### API Calling

static std::shared_ptr<Preprocessor> hilens::Preprocessor::Create()

### Return Value

If the operation is successful, the pointer of a preprocessor is returned. Otherwise, nullptr is returned.

## 7.3 Changing the Image Size

This section describes how to resize an image.

## API Calling

HiLensEC hilens::Preprocessor::Resize(const cv::Mat & src, cv::Mat & dst, unsigned int w, unsigned int h, int type = 0)

## Parameter Description

**Table 7-1** Parameters

| Parameter | Description |
| --- | --- |
| src | Source image, which must be in the NV21 format. Width range: [64, 1920], a multiple of 2; Height range: [64, 1080], a multiple of 2.<br>If the input is not in the NV21 format, convert the input source image to the NV21 format. Refer to **Converting the Image Color Format** for details. |
| dst | Target image |
| w | Width of the image after zoom-in/out. Value range: [64, 1920], a multiple of 2. |
| h | Height of the image after zoom-in/out. Value range: [64, 1080], a multiple of 2. |
| type | Format of the target image. The value **0** indicates NV21, and value **1** indicates NV12. The default value is **0**. |

## Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

# 7.4 Cropping Images

This section describes how to crop an image.

## API Calling

HiLensEC hilens::Preprocessor::Crop(const cv::Mat & src, cv::Mat & dst, unsigned int x, unsigned int y, unsigned int w, unsigned int h, int type = 0)

### Parameter Description

**Table 7-2**

| Parameter | Description |
| --- | --- |
| src | Source image, which must be in the NV21 format. Width range: [64, 1920], a multiple of 2; Height range: [64, 1080], a multiple of 2.<br>If the input is not in the NV21 format, convert the input source image to the NV21 format. Refer to **Converting the Image Color Format** for details. |
| dst | Target image |
| x | X coordinate in the upper left corner of the cropped area. Value range: [0, 1920], a multiple of 2. |
| y | Y coordinate in the upper left corner of the cropped area. Value range: [0, 1080], a multiple of 2. |
| w | Width of the image after zoom-in/out. Value range: [64, 1920], a multiple of 2. |
| h | Height of the image after zoom-in/out. Value range: [64, 1080], a multiple of 2. |
| type | Format of the target image. The value **0** indicates NV21, and value **1** indicates NV12. The default value is **0**. |

### Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

## 7.5 Converting the Image Color Format

This section describes how to convert the color format of an image. The native OpenCV does not provide the option of converting RGB/BGR to NV12/NV21.

### API Calling

HiLensEC hilens::CvtColor(const cv::Mat & src, cv::Mat & dst, CvtCode code)

## Parameter Description

**Table 7-3** Parameters

| Parameter | Description |
| --- | --- |
| src | Source image (BGR888 or RGB888). |
| dst | Target image |
| code | Color conversion code, which specifies the conversion type. The options are RGB2YUV_NV12, RGB2YUV_NV21, BGR2YUV_NV12, and BGR2YUV_NV21.<br>`enum hilens::CvtCode`<br>For details about the enumerated values, see **Table 7-4**. |

**Table 7-4** Color conversion code

| Enumerated Value | Description |
| --- | --- |
| BGR2YUV_NV12 | Convert BGR to YUV_NV12. |
| RGB2YUV_NV12 | Convert RGB to YUV_NV12. |
| BGR2YUV_NV21 | Convert BGR to YUV_NV21. |
| RGB2YUV_NV21 | Convert RGB to YUV_NV21. |

## Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

# 8 Model Management

## 8.1 Introduction

### hilens::Model Class

The model manager is used to load the model and perform inference.

```
#include <model.h>
```

### Destructor

```
~Model()
virtual hilens::Model::~Model( )
```

When a model is destructed, resources such as hiai::Graph are released.

## 8.2 Creating Models

This section describes how to construct a model. HiLens Kit can use the models supported by the Ascend 310 for inference. Use this method to construct a model for subsequent inference. If the model fails to be constructed, the system displays CreateError and an error code is recorded in the log (for example, 0x1013011 indicates that the model path is incorrect). When the returned object is destructed, the corresponding model resource is released.

### API Calling

static std::shared_ptr<Model> hilens::Model::Create(const std::string & filename)

### Parameter Description

**Table 8-1** Parameters

| Parameter | Description |
| --- | --- |
| filename | Path of the model file. If the model is stored in **./mymodels/test.om**, the file name is **./mymodels/test.om**. |

### Return Value

If the operation is successful, the pointer of the model manager instance is returned. If the operation fails, nullptr is returned.

## 8.3 Model Inference

This section describes how to input data to a model for inference. After the inference is complete, the inference result is returned.

### API Calling

virtual HiLensEC hilens::Model::Infer(const InferDataVec & inputs, InferDataVec & outputs)

### Parameter Description

**Table 8-2** Parameters

| Parameter | Type | Description |
| --- | --- | --- |
| inputs | InferDataVector. Refer to **Parameter Type** for details. | Input inference data |
| outputs | InferDataVector. Refer to **Parameter Type** for details. | Inference output |

### Parameter Type

- InferDataVec

  Model inference input and output

  ```
  typedef std::vector<InferData> hilens::InferDataVec
  ```

- InferData

  ```
  struct InferData
  {
  unsigned int size; // Output size
  std::shared_ptr<unsigned char> data; // Data pointer
      /**
  ```

```
* @brief: Construct an empty model for inference.
     */
    InferData() : size(0), data(nullptr) {}

    /**
* @brief: Construct the InferData from a cv::Mat.
* @param img Input images.
     */
    InferData(const cv::Mat &img);

    /**
* @brief: Construct an InferData from a group of pointer data.
* @param data: The data pointer. This constructor copies the data.
* @param size: Data size (byte)
     */
    InferData(const unsigned char *data, unsigned int size);
};
```

## Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

If the actual inference input is different from the model input, the inference will fail. In this case, the return value of inference will be an int error code, and error information will be recorded in logs. You can locate the error based on the error information.

# 9 Output

## 9.1 Introduction

### hilens::Display Class

This section describes how to use the Display class to output images to a monitor.

```
#include <output.h>
```

### Constructor and Destructor

```
~Display()
virtual hilens::Display::~Display()
```

## 9.2 Constructing Output Monitors

This section describes how to construct monitors to display images or output images to video streams. If the operation fails, the system displays "CreateError". You can view the skill logs or output to locate the error cause.

If the file type is H264_FILE, the generated file is only an H.264-encoded raw video stream and does not contain information such as the frame rate. In addition, the HiLens Framework does not limit the file size. Therefore, it is recommended that this function be used only for debugging.

### API Calling

static std::shared_ptr hilens::Display::Create(Type type, const char * path = NULL)

### Parameter Description

**Table 9-1** Parameters

| Parameter | Description |
| --- | --- |
| type | Display type. The options are HDMI, RTMP, and H264_FILE. |
| path | If the file type is HDMI, ignore this parameter. If the file type is RTMP, set the path to the RTMP server URL (**rtmp:*//xxxx***). If the file type is H264_FILE, set the path to the output file path (for example, **hilens::GetWorkspacePath()+"/out.h264"**). |

### Return Value

If the operation is successful, a monitor instance is returned. If the operation fails, nullptr is returned.

## 9.3 Outputting a Frame of Image

This section describes how to output a frame of image. When the following API is called for the first time, the output module sets the video size based on the input image size. In the subsequent calling, the skill must ensure that the input image size is the same as the previous one.

### API Calling

virtual HiLensEC hilens::Display::Show(const cv::Mat & frame)

### Parameter Description

**Table 9-2** Parameters

| Parameter | Description |
| --- | --- |
| frame | Image to be displayed. The image must be in the NV21 format. |

### Return Value

If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

# 9.4 Upload Files

## UploadFile()

This section describes how to upload files to OBS. During the file upload, other threads are blocked (in the waiting state). They are unblocked until the files are uploaded successfully. You can configure the target OBS bucket on the HiLens console. For details, see the "Configuring the Data Storage Location" section in *HiLens User Guide*.

- **API calling**

  HiLensEC hilens::UploadFile(const std::string & key, const std::string & filepath, const std::string & mode)

- **Parameter description**

  **Table 9-3** Parameters

| Parameter | Description |
| --- | --- |
| key | Name of the file uploaded to OBS |
| filepath | Absolute path of the file to be uploaded |
| mode | Upload mode. The options are **write** (overwrite) and **append** (add). |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

## UploadFileAsync()

The following API is used to asynchronously upload files. A response is returned immediately after a file is uploaded successfully.

- **API calling**

  HiLensEC hilens::UploadFileAsync(const std::string & key, const std::string & filepath, const std::string & mode, void(*)(int) callback = NULL)

- **Parameter description**

  **Table 9-4** Parameters

| Parameter | Description |
| --- | --- |
| key | Name of the file uploaded to OBS |
| filepath | Absolute path of the file to be uploaded |

| Parameter | Description |
| --- | --- |
| mode | Upload mode. The options are **write** and **append**. |
| callback | Callback function |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

# 9.5 Uploading Buffer Data

## UploadBuffer()

This section describes how to upload buffer data to OBS. During the buffer upload, threads are blocked. They are not blocked until the buffer data is uploaded successfully. You can configure the target OBS bucket on the HiLens console. For details, see the "Configuring the Data Storage Location" section in *HiLens User Guide*.

- **API calling**

  HiLensEC hilens::UploadBuffer(const std::string & key, const unsigned char * buffer, const unsigned char * buffer, size_t bufferSize, const std::string & mode)

- **Parameter description**

  **Table 9-5** Parameters

| Parameter | Description |
| --- | --- |
| key | Name of the file uploaded to OBS |
| buffer | Pointer of the buffer to be uploaded |
| bufferSize | Size of the buffer to be uploaded |
| mode | Upload mode. The options are **write** and **append**. |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

## UploadBufferAsync()

The following API is used to asynchronously upload the buffer data. A response is returned immediately after the buffer data is uploaded successfully.

- **API calling**

HiLensEC hilens::UploadBufferAsync(const std::string & key, std::shared_ptr<const unsigned char> buffer, size_t bufferSize, const std::string & mode, void(*)(int) callback = NULL)

- **Parameter description**

  **Table 9-6** Parameters

| Parameter | Description |
| --- | --- |
| key | Name of the file uploaded to OBS |
| buffer | Pointer of the buffer to be uploaded |
| bufferSize | Size of the buffer to be uploaded |
| mode | Upload mode. The options are **write** and **append**. |
| callback | Callback function |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

# 9.6 Sending POST Requests

This section describes how to send POST requests synchronously. During the request sending process, the threads are blocked until the requests are sent successfully. The TLS1.2 security protocol is supported, and the timeout period is set to 20 seconds.

## API Calling

int hilens::POST(const std::string & url, const Json::Value & body, long & httpcode, std::string * response = NULL, POSTHeaders * headers = NULL)

## Parameter Description

**Table 9-7** Parameters

| Parameter | Description |
| --- | --- |
| url | Uniform resource locator |
| body | JSON object of the message body |
| httpcode | The return value of the HTTP request. If the Return Value is **200**, the request is sent successfully. If the return value is **404**, the requested page does not exist. |

| Parameter | Description |
| --- | --- |
| response | Response. This parameter can be left blank. |
| headers | Request header. If this parameter is left blank, the header is ignored.<br>typedef std::vector<std::string> hilens::POSTHeaders<br>The header of a called POST request can be transferred as a POST parameter (for example, headers.push_back("Content-Type: application/json"). |

### Return Value

Return value of cURL. The value **0** indicates the operation is successful.

## 9.7 Sending Messages

### SendMessage()

This section describes how to send messages synchronously. During the message sending process, other threads are blocked (in the waiting state). They are unblocked until the messages are sent successfully. You need to configure message subscription on the console first. For details, see the "Subscribing to Messages" section in *HiLens User Guide*. Only 1.0.7 to 1.2.2 versions are supported.

- **API calling**

  HiLensEC hilens::UploadBuffer(const std::string & subject, const std::string & message)

- **Parameter description**

  **Table 9-8** Parameters

  | Parameter | Description |
  | --- | --- |
  | subject | Email subject. This parameter is valid only when **Email** is set. The value can contain up to 170 characters. |
  | message | Message content, which contains up to 85 characters. |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

### SendMessageAsync()

This section describes how to send messages asynchronously. You need to configure message subscription on the console first. For details, see the "Subscribing to Messages" section in *HiLens User Guide*. Only 1.0.7 to 1.2.2 versions are supported.

- **API calling**

  HiLensEC hilens::UploadBuffer(const std::string & subject, const std::string & message, void (*callback)(int) = NULL)

- **Parameter description**

  **Table 9-9** Parameters

| Parameter | Description |
| --- | --- |
| subject | Email subject. This parameter is valid only when **Email** is set. The value can contain up to 170 characters. |
| message | Message content, which contains up to 85 characters |
| callback | Callback function |

- **Return values**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

## 9.8 Playing an Audio File

This section describes how to play a local audio file in AAC format. Connect a headset or sound box to the audio output port of the HiLens Kit device. You can hear the sound calling this API.

- **API calling**

  HiLensEC PlayAacFile(const std::string filePath, int vol)

- **Parameter description**

  **Table 9-10** Parameters

| Parameter | Description |
| --- | --- |
| filePath | Absolute path of the local audio file. Chinese characters are not supported. |
| vol | Volume of the audio to be played. The value range is [-121, 6]. |

NOTE

This API and the API in **Constructing Audio Collectors** cannot be called at the same time.

- **Return Value**

  If the return value is **0**, the operation is successful. Otherwise, the operation fails. Refer to **Error Codes** for details about the failure response parameters.

# 10 Resource Management

## 10.1 Obtaining Model Paths

This section describes how to return the path of a skill model directory.

If the skill code package is separated from the model, the model will be downloaded to a specified directory. The following API is used to obtain the path. If the HiLens Framework does not obtain the directory where the model is located, the current path is returned.

### API Calling

std::string hilens::GetModelDirPath()

### Return Value

A character string is returned, indicating the path of the skill model directory (with a slash (/) at the end). If the path of the skill model directory fails to be obtained, a null character string is returned.

## 10.2 Obtaining the Directories of the Skill Workspaces

This section describes how to return the path of a skill workspace directory.

Due to problems such as certificate verification, writing operations are not allowed in the skill installation directory. Therefore, you need to specify the writable workspace of each skill.

### API Calling

std::string hilens::GetWorkspacePath()

### Return Value

If the operation is successful, the path of a skill workspace (with a slash (/) at the end) is returned. Otherwise, a null character string is returned.

# 10.3 Obtaining Skill Configurations

This section describes how to obtain the skill configurations, which are the JSON objects (jsoncpp) parsed from the content in the skill configuration files. Note that GetSkillConfig() reads the configuration file and parses it into a JSON object each time. Therefore, if you need to read multiple configuration items, save the return values as variables and do not frequently invoke GetSkillConfig().

## API Calling

Json::Value hilens::GetSkillConfig()

## Return Value

The JSON object of the skill configurations is returned. If the parsing fails, an empty **JSON::Value** is returned. You can use **.empty()** to check whether the **JSON::Value** is empty.

# 10.4 Downloading Files

This section describes how to download files from OBS.

## API Calling

HiLensEC hilens::DownloadFileFromOBS(const std::string & url, const std::string & downloadTo)

## Parameter Description

**Table 10-1** Parameters

| Parameter | Description |
| --- | --- |
| url | The URL of the OBS resource you want to download. For details on how to obtain the URL, see **Accessing an Object Using Its URL**. |
| download_to | The directory where you want to download the file to |

## Return Values

If the return value is **0**, the operation is successful. Otherwise, the operation fails. For details about the failure response parameters, see **Error Codes**.

# 10.5 Calculating the MD5 Values of Files

This section describes how to calculate the MD5 values of files.

## API Calling

std::string hilens::MD5ofFile(const std::string & filepath)

## Parameter Description

**Table 10-2** Parameters

| Parameter | Description |
| --- | --- |
| file | The file path. |

## Return Value

MD5 value of a file. The value is a character string. If the file fails to be read, a null character string is returned.

# 10.6 Example - Resource Management

The following is an example of resource management:

```
#include <cstdio>
#include <hilens.h>
#include <string>

using namespace hilens;
using namespace cv;

void ResourceManage() {
  // Obtain the skill workspace path (with a slash (/) at the end).
  auto skill_path = hilens::GetWorkspacePath();

  // Obtain the skill model path (with a slash (/) at the end).
  auto model_path = hilens::GetModelDirPath();

  // Obtain the skill configuration. If no information is obtained, None is returned.
  auto skill_config = hilens::GetSkillConfig();
  // Assume that face_dataset is a skill configuration item and its value is the directory of face_dataset.zip
in OBS.
  // Configure the skill by referring to User Guide.
  auto face_dataset_url = skill_config["face_dataset"]["value"].asString();
  // Download the file from OBS to the skill workspace directory, and check whether the file is successfully
downloaded based on the return value.
  auto ret =
      hilens::DownloadFileFromOBS(face_dataset_url, hilens::GetWorkspacePath());
  if (ret != hilens::OK) {
    hilens::Error("Failed to download from obs");
  }

  // Create a folder in the skill workspace directory and decompress the downloaded file.
  std::string cmd = "mkdir " + hilens::GetWorkspacePath() + "face_dataset";
  auto result = system(cmd.c_str());
```

```
  if (result != 0) {
    hilens::Error("Failed to mkdir");
  }

  cmd = "unzip " + hilens::GetWorkspacePath() + "face_dataset.zip  -d " +
        hilens::GetWorkspacePath() + "face_dataset/";
  result = system(cmd.c_str());
  if (result != 0) {
    hilens::Error("Failed to unzip");
  }
}

int main() {
  auto ret = hilens::Init("hello");
  if (ret != hilens::OK) {
    hilens::Error("Failed to init");
    return -1;
  }

  ResourceManage();

  hilens::Terminate();
  return 0;
}
```

# 11 Hard Example Upload

## 11.1 Introduction to Hard Example Upload

The 1.1.2 version supports hard example mining algorithms for edge AI. If you want to use the hard example upload API, upgrade the firmware version to 1.1.2. For details about how to upgrade the firmware, see **Upgrading HiLens_Device_Agent Firmware**.

The following hard example mining algorithms are supported:

- **Image classification**

  CrossEntropyFilter(threshold_cross_entropy)

  Principle: Determine whether the entropy is less than the cross entropy of the inference result. If the entropy is less than the cross entropy, the sample is a hard example.

  Input: **prediction classes list**, for example, **[class1-score, class2-score, class2-score,....]**, where **class-score** indicates the class score ranging from 0 to 1.

  Output: **True** or **False**. **True** indicates that the image is a hard example, and **False** indicates the image is not a hard example.

- **Object detection**

  IBT (image-box-thresholds)

  Principle: **box_threshold** calculates the hard example coefficient (the confidence score less than the threshold to the total number of output inference boxes). **img_threshold** determines whether an image is a hard example.

  Input: **prediction boxes list**, for example, **[bbox1, bbox2, bbox3,....]**, where **bbox = [xmin, ymin, xmax, ymax, score, label]**, **x**, and **y** indicate the coordinates of the bounding box, **score** indicates the confidence score ranging from 0 to 1, and **label** indicates the class label.

  Output: **True** or **False**. **True** indicates that the image is a hard example, and **False** indicates the image is not a hard example.

  CSF(confidence score filter)

  Principle: **box_threshold_low** and **box_threshold_up** determine whether an image is a hard example. As long as the confidence score of an output box is

within the range specified by **[box_threshold_low, box_threshold_up]**, the image is a hard example.

Input: **prediction boxes list**, for example, **[bbox1, bbox2, bbox3,....]**, where **bbox = [xmin, ymin, xmax, ymax, score, label]**, **x**, and **y** indicate the coordinates of the bounding box, **score** indicates the confidence score ranging from 0 to 1, and **label** indicates the class label.

Output: **True** or **False**. **True** indicates that the image is a hard example, and **False** indicates the image is not a hard example.

# 11.2 Constructing a HardSample Instance

## API Calling

HardSampleInferface &hilens::GetHardSampleInstance()

## Return Value

**HardSampleInferface** object

# 11.3 Initializing

This section describes how to initialize the **HardSampleInferface** object.

## API Calling

virtual bool hilens::HardSampleInferface::Init(const float thresholdOne, const float thresholdTwo, const DetectionFilterType filterType)

**Table 11-1** Parameters

| Parameter | Description |
| --- | --- |
| thresholdOne | Threshold |
| thresholdTwo | Threshold |

| Parameter | Description |
| --- | --- |
| filterType | Type of the hard example filter. The value can be **CrossEntropyFilter**, **IBT**, or **CSF**. For details, see **Introduction to Hard Example Upload**.<br>If **CrossEntropyFilter** is used, **thresholdOne** corresponds to **threshold_cross_entropy** of **CrossEntropyFilter**, and **thresholdTwo** can be any value.<br>If **IBT** is used, **thresholdOne** and **thresholdTwo** correspond to **box_threshold** and **img_threshold** of **IBT**, respectively.<br>If **CSF** is used, **thresholdOne** and **thresholdTwo** correspond to **box_threshold_low** and **box_threshold_up** of **CSF**, respectively. |

### Return Value

A bool value is returned, indicating whether the initialization is successful or fails.

## 11.4 Determining a Hard Example Image

This section describes how to determine whether an input image is a hard example.

### API Calling

virtual bool Filter(const float inferResult[], const int size);

virtual bool Filter(const std::vector<Bbox> &bboxList, DetectionFilterType type);

**Table 11-2** Parameter description 1

| Parameter | Description |
| --- | --- |
| inferResult[] | Float array, which contains the confidence score of each class obtained by the classification algorithm |
| size | Input size |

**Table 11-3** Parameter description 2

| Parameter | Description |
| --- | --- |
| bboxList | std::vector<Bbox>. The Bbox structure is defined as follows:<br>structBbox<br>{<br>float xmin;<br>float ymin;<br>float xmax;<br>float ymax;<br>float score;<br>int label;<br>Bbox(floatbboxXmin, floatbboxYmin, floatbboxXmax, floatbboxYmax, floatbboxScore, intbboxLabel):<br>xmin(bboxXmin), ymin(bboxYmin), xmax(bboxXmax), ymax(bboxYmax), score(bboxScore), label(bboxLabel){}<br>};<br>**Table 11-4** describes related parameters. |
| type | Type of the hard example filter. The value can be **CrossEntropyFilter**, **IBT**, or **CSF**. For details, see **Introduction to Hard Example Upload**. |

**Table 11-4** Parameters in the Bbox structure

| Value | Description |
| --- | --- |
| xmin | Coordinates of a bounding box |
| ymin | Coordinates of a bounding box |
| xmax | Coordinates of a bounding box |
| ymax | Coordinates of a bounding box |
| score | Score of a bounding box |
| label | Class of a bounding box |

### Return Value

A bool value is returned, indicating whether the image is a hard example.

## 11.5 Obtaining Hard Example Configurations

### API Calling

Json::Value GetHardSampleConfig()

### Return Value

**Json::Value** object, which can be used to parse the value of each configuration item

## 11.6 Updating Hard Example Configurations

This section describes how to update hard example configurations in the hard example configuration file and update the hard example upload status on the cloud based on the input.

### API Calling

HiLensEC SetHardSampleConfig(const std::string &confStr)

**Table 11-5** Parameters

| Parameter | Description |
| --- | --- |
| confStr | Hard example configuration to be updated. The value must be a JSON string. |

### Return Value

HiLensEC error code. If the operation is successful, **0** is returned. Otherwise, the operation fails.

# 12 Log

## 12.1 Setting the Log Levels

This section describes how to set the log levels.

### API Calling

void hilens::SetLogLevel(LogLevel level)

### Parameter Description

**Table 12-1** Parameters

| Parameter | Description |
| --- | --- |
| level | Log level. The value can be **Trace**, **Debug**, **Info**, **Warning**, **Error**, or **Fatal**.<br>enum hilens::LogLevel<br>Refer to **Table 12-2** for the enumerated values. |

**Table 12-2** Enumerated Value

| Log Level | Description |
| --- | --- |
| TRACE | Logs of the Trace level or higher |
| DEBUG | Logs of the Debug level or higher |
| INFO | Logs of the Info level or higher |
| WARNING | Logs of the Warning level or higher |
| ERROR | Logs of the Error level or higher |
| FATAL | Logs of the Fatal level |

### Return Value

None

## 12.2 Trace Logs

This section describes logs of the Trace level. The log usage is similar to that of printf.

### API Calling

void hilens::Trace(const char * fmt, ... )

### Parameter Description

**Table 12-3** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. The usages of the additional parameters are similar to that of printf. For example, hilens::Trace (The character string is %s \n", str). |

### Return Value

None

## 12.3 Debug Logs

This section describes the logs of the Debug level. The log usage is similar to that of printf.

## API Calling

void hilens::Debug(const char * fmt, ... )

## Parameter Description

**Table 12-4** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. For example, hilens::Debug (The character string is %s \n", str). |

## Return Value

None

# 12.4 Info Logs

This section describes the logs of the Info level. The log usage is similar to that of printf.

## API Calling

void hilens::Info(const char * fmt, ... )

### Parameter Description

**Table 12-5** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. For example, hilens::Info (The character string is %s \n", str). |

### Return Value

None

## 12.5 Warning Logs

This section describes the logs of the Warning level. The log usage is similar to that of printf.

### API Calling

void hilens::Warning(const char * fmt, ... )

### Parameter Description

**Table 12-6** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |

| Parameter | Description |
| --- | --- |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. For example: hilens::Warning (The character string is %s \n", str). |

### Return Value

None

## 12.6 Error Logs

This section describes the logs of the Error level. The log usage is similar to that of printf.

### API Calling

void hilens::Error(const char * fmt, ... )

### Parameter Description

**Table 12-7** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. For example, hilens::Error (The character string is %s \n", str). |

### Return Value

None

## 12.7 Fatal Logs

This section describes the logs of the Fatal level. The log usage is similar to that of printf.

### API Calling

void hilens::Fatal(const char * fmt, ... )

### Parameter Description

**Table 12-8** Parameters

| Parameter | Description |
| --- | --- |
| fmt | A string that can contain embedded formatting tags that can be replaced by the values specified in the subsequent additional parameters and formatted as required. A log record can contain a maximum of 255 characters. |
| ... (Additional parameters) | Depending on the **fmt** string, the function may require a series of additional parameters. Each parameter contains a value to be inserted and replaces each % tag specified in the **fmt** parameter. The number of parameters must be the same as the number of % tags. For example, hilens::Fatal (The character string is %s \n", str). |

### Return Value

None

# 13 Error Codes

The error codes returned by the HiLens Framework are of the HiLens EC type. **Table 13-1** describes the error codes (HiLens EC enumerated values).

**Table 13-1** Error codes

| Error Code | Description |
| --- | --- |
| OK=0 | No error. |
| UNKNOWN_ERROR | Unknown error. |
| INIT_CURL_ERROR | CURL initialization error. |
| CREATE_DIR_FAILED | Failed to create folders. |
| OPENFILE_FAILED | Failed to open files. |
| RENAME_FAILED | Renaming failed. |
| ACCESS_FILE_FAILED | The file does not exist or you do not have the permissions to access the files. |
| INVALID_BUF | Invalid buffer. |
| COULDNT_RESOLVE_HOST | Parsing failed. Check whether the network connection is normal. |
| WRITE_ERROR | Write error. Check whether you have the write permission on the downloaded directory and whether the space is sufficient. |
| TIMEOUT | Request timed out. |
| AUTH_FAILED | Incorrect authentication information. Check whether the AK, SK, and token are valid. |
| NOT_FOUND | The object does not exist. |

| Error Code | Description |
| --- | --- |
| SERVER_ERROR | Internal service error. |
| OBJECT_CONFLICT | Object conflict. |
| APPEND_FAILED | Appending failed (for example, appending to an object that cannot be appended). |
| HIAI_SEND_DATA_FAILED | HiAI engine fails to send data. Analyze the fault based on logs. |
| HIAI_INFER_ERROR | HiAI engine inference error. Analyze the fault based on logs. The possible cause is that the actual input size does not match the input size of the model. |
| INVALID_SRC_SIZE | During image processing, the src size does not meet the restriction conditions. |
| INVALID_DST_SIZE | During image processing, the dst size does not meet the restriction conditions. |
| MPP_PROCESS_FAILED | The MPP fails to process images. |
| WEBSOCKET_ERROR | WebSocket error. |
| CONFIG_FILE_ERROR | Incorrect configuration file. |
| INVALID_PARAM | Incorrect parameter. |
| INIT_LOG_ERROR | Log initialization failed. |
| INIT_MIC_ERROR | Microphone initialization failed. |
| INIT_AENC_ERROR | Audio encoding initialization failed. |
| INIT_ADEC_ERROR | Audio decoding initialization failed. |
| INIT_AO_ERROR | Audio output initialization failed. |
| AUDIO_CHECK_ERROR | The audio file is not supported. |
| AUDIO_SYSTEM_INIT_FAILED | Audio system initialization failed. |