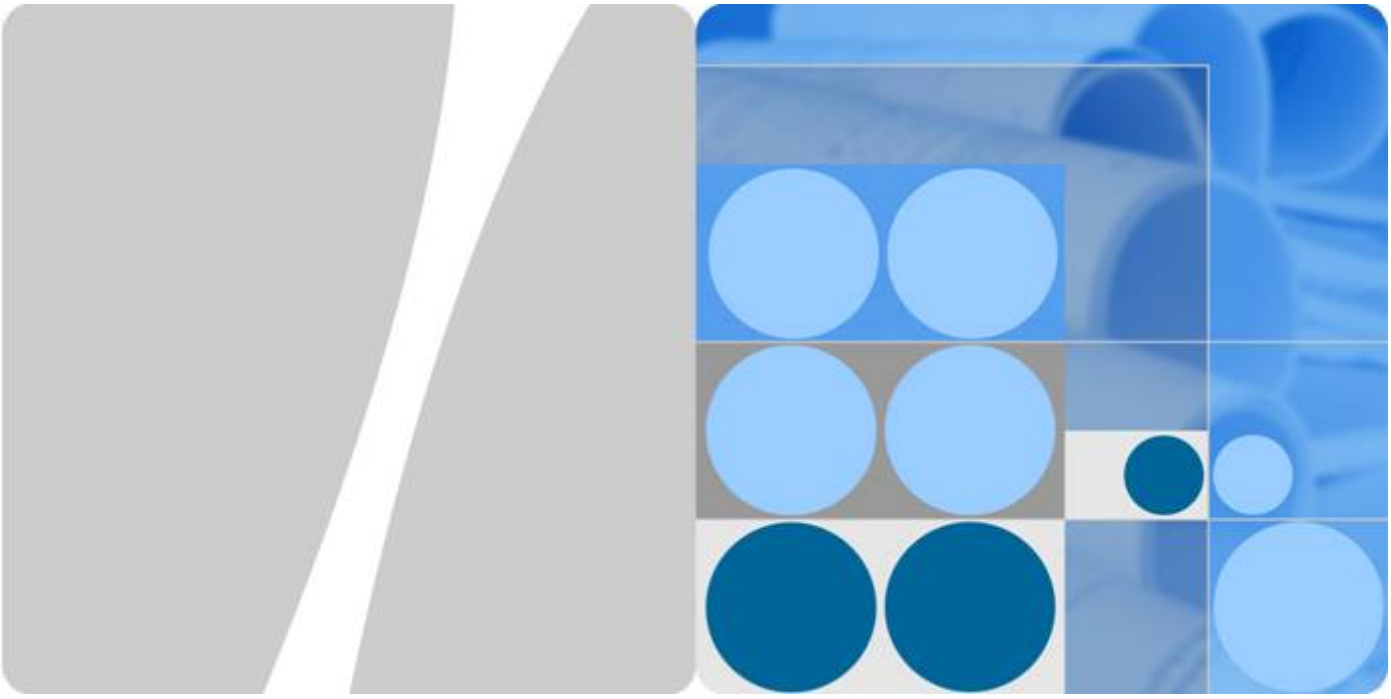

Huawei HiLens

Developer Guide

Issue 01

Date 2020-03-27

HUAWEI TECHNOLOGIES CO., LTD.

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

When developing skills on the Huawei HiLens management console, you need to edit or upload logic code online. The HiLens Framework is required in the logic code. This document describes how to use HiLens Framework APIs in the logic code when developing skills that can run on HiLens Kit devices. You can search for the required content based on **Table 1-1**.

Table 1-1 Before you start

Chapter	Description
HiLens Framework Introduction	Describes HiLens Framework API constituents and API list.
Initialization **Video Input Module** **Preprocessing Module** **Model Management Module** **Output Module** **Resource Management Module** **Log Module**	Details the classes and functions encapsulated in the HiLens Framework.
Change History	Records the document change history.

2 HiLens Framework Introduction

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

Figure 2-1 shows the layered structure of the HiLens Framework. The HiLens Framework encapsulates the bottom-layer multimedia processing libraries (cameras/microphone driver module Media_mini), the image processing library (DVPP) related to D chips, and the model management library (ModelManager). You can also use the familiar visual processing library OpenCV. In addition, the HiLens Framework provides the following modules for you to develop skills, such as human figure detection, facial recognition, and fatigue driving detection. **Table 2-1** details the modules.

Figure 2-1 HiLens Framework

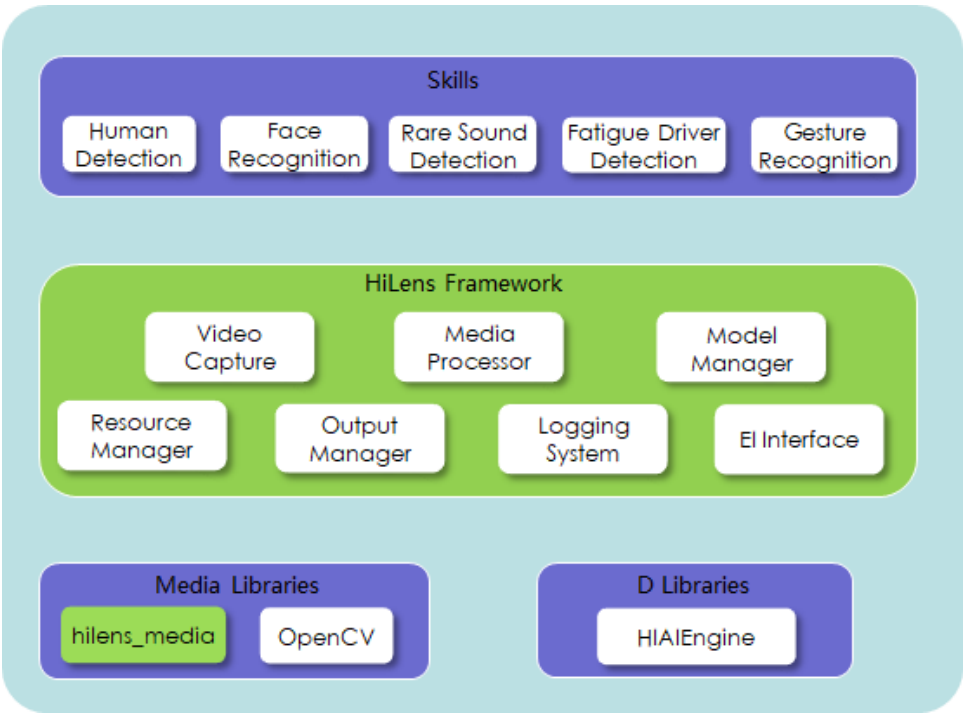

Table 2-1 Module description

No.	Module	Function
1	Input Manager	Input module: manages the access of input data such as video and audio data.
2	Media Processor	Preprocessing module: processes media data such as video and audio data.
3	Model Manager	Model management module: initializes models and infers based on the models.
4	Output Manager	Output module: manages output tasks such as streams, files, and message notifications.
5	Resource Manager	Resource management module: manages paths of resources such as files, images, and models.
6	Logging System	Log module: manages the log system.

3 Initialization

3.1 Initializing the HiLens Framework

This API is used to initialize the HiLens Framework. Before calling other APIs of the HiLens Framework, you need to perform global initialization.

- **API calling**

  hilens.init(verify)

- **Parameter description**

  Table 3-1 Parameters

Parameter	Mandatory	Type	Description
verify	Yes	Character string	The value is a string of 0 to 128 characters. The value must be the same as the verification value in the basic information entered when you create a skill on the Huawei HiLens management console. If they are different, the HiLens Framework forcibly stops the skill.

- **Return value**

  The value **0** indicates that the HiLens Framework is initialized successfully. Other values indicate that the HiLens Framework fails to be initialized.

  This method can also be used to verify whether the skill is corrupted or tampered with. To use this function, the **verify** parameter must be the return value of a function compiled by the developer. The return value is the hash value of the file to be verified in real time. After developing a skill, the developer uses the same hash method to calculate the hash value and enters the verification value when creating the skill on the console. The following is an example:


```
#! /usr/bin/python3.7

import hilens
def verify():
    # You need to implement a method to verify the program identity (to prevent the program from being damaged or tampered with).
    # For example, if the hash value of an important file in a skill package can be calculated, verify must return a string of 1 to 128 bytes.
    # On the HiLens platform, enter the hash value during skill development. After the init method is invoked, the skill automatically sends the hash value to the platform for comparison and verifies the skill license.

    # During debugging, you can use a fixed character string for verification to facilitate code modification.
    # The source code of Python scripts is easy to be tampered with when being delivered to devices. For commercial skills, you are advised to use C++ for development.
    # Note: Do not use hard-coded character strings to verify skills that are officially released.
    return "hello"

def main():
     # Initialize HiLens.
    rc = hilens.init(verify())
# If you do not want to use this function in skill development and debugging phase, enter a static character string.
    # Example: hilens.init("hello")
    if rc != 0:
        hilens.error("Failed to initialize HiLens")
        return

   # Business code
    pass

    # Clear resources.
    hilens.terminate()

if __name__ == '__main__':
    main()
```

3.2 Releasing HiLens Framework Resources

This API is used to terminate the HiLens Framework. When the program ends, this API is called to release related resources.

- **API calling**

  hilens.terminate()

- **Return value**

  The value 0 indicates that the resources are released successfully. Other values indicate that the resources fail to be released.

4 Video Input Module

4.1 Video Collector

This API is used to construct a video collector to open the HiLens Kit camera, construct an IP camera video collector (for the IP cameras that use the RTSP protocol), or construct a UV camera video collector (for the cameras that comply with the USB video class specifications). Currently, the HiLens Kit has two USB ports, but only one USB camera can be connected.

After the firmware is upgraded to 1.0.7 or later, this API can be used to read local MP4 files and set the width and height of the video frames read from the IP camera or local MP4 files.

- **API calling**

  hilens.VideoCapture(camera)

  1.0.7 and later firmware versions:

  hilens.VideoCapture(camera, width, height)

- **Parameter description**

Table 4-1 Parameters

Parameter	Mandatory	Type	Description
camera	No	• Character string • Integer 0	• If no parameter is entered, a video collector is constructed to open the HiLens Kit camera. Only one skill of a device can use the camera. Otherwise, resource preemption will occur, causing an error. • If the camera name in the device configuration file is entered, an IP camera video collector is constructed. 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*. To view the camera name, log in to the Huawei HiLens console and choose **Device Management** > **HiLens Kit** > **Camera Configuration** > **Camera Management**. • If the integer 0 is entered, a UV camera video collector is constructed (the UV camera needs to be inserted). • If the path of the local MP4 video file is entered, an MP4 video collector is constructed.
width	No. This parameter must be used together with **height**.	Integer	Set the width of the read video frame image. The value must be a multiple of 16. The recommended value is a multiple of 32. The minimum value is **128**. This parameter can be set only for IP cameras and MP4 video files. If you do not set this parameter, the original width and height of the video frame are used by default.
height	No. This parameter must be used together with **width**.	Integer	Set the height of the read video frame image. The value must be a multiple of 2. The minimum value is **128**. This parameter can be set only for IP cameras and MP4 video files. If you do not set this parameter, the original width and height of the video frame are used by default.

- **Return value**
  - Video collector with a camera.
  - IP camera video collector
  - UV camera video collector
  - MP4 video collector
  - If the construction fails, "CreateError" is reported. You can view skill logs.

4.2 Reading Video Frames from Cameras

This API is used to read a video frame. Note that the IP camera and MP4 video return the **YUV_NV21** color data, while the UV camera returns the BGR color data.

- **API calling**

  hilens.VideoCapture.read()
- **Return Value**

  One frame of video data. The parameter type is NumPy array (**dtype** is **uint8**), which is compatible with CV2.

4.3 Obtaining the Video Width

This API is used to obtain the video width.

- **API calling**

  hilens.VideoCapture.width
- **Return value**

  Video width

4.4 Obtaining the Video Height

This API is used to obtain the video height.

- **API calling**

  hilens.VideoCapture.height
- **Return value**

  Video height

4.5 Example - Input

The following is an example of the input module:

NOTE

- The following example calls the video collector API when the firmware version is 1.0.7 or later. You can set the video size when calling this API. That is, you can set the **width** and **height** parameters when calling **hilens.VideoCapture(camera, width, height)**.
- When the HiLens Studio or firmware version is earlier than 1.0.7, the video size cannot be set.


```
#! /usr/bin/python3.7

import hilens
import numpy as np

def run():
    # Construct a camera.
    cap0 = hilens.VideoCapture()            # Built-in camera
     cap1 = hilens.VideoCapture("IPC1")      # IP camera whose name is IPC1 in the camera configuration file. To configure the camera, log in to the Huawei HiLens console, choose Skill Development > Skill Management > Create Skill, and add the configuration in the Runtime Configuration area.
    cap2 = hilens.VideoCapture("rtsp://192.168.1.1/video") # RTSP video stream whose address is rtsp://192.168.1.1/video
    cap3 = hilens.VideoCapture("/tmp/test.mp4", 1920, 1080) # Read the test.mp4 video file in the /tmp directory on HiLens Kit cameras and adjust the video frame width and height to 1920 and 1080. (The firmware version must be 1.0.7 or later.)
    cap4 = hilens.VideoCapture(0)           # Currently, only one UV camera is supported. The camera ID is 0.

    # Obtain the video size.
    w = cap0.width
    h = cap0.height
    hilens.info("width: %d, height: %d" % (w, h))

    # Read video data.
    frame0 = cap0.read()
    # Other processing
    pass

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


5 Audio Input Module

5.1 Audio Collector

This API is used to construct an audio collector to obtain audio data from local microphones or audio files.

- **API calling**

  1.0.8 or later

  hilens.AudioCapture(file_path)

  1.1.2 or later

  hilens.AudioCapture(sample_rate, bit_width, nSamples, sound_mode)

- **Parameter description**

  Table 5-1 Parameters

Parameter	Mandatory	Type	Description
file_path	No	Character string	Audio file path. Audio data is obtained from the file to construct a collector for audio file data.

Parameter	Mandatory	Type	Description
sample_rate	No	Integer	Sampling rate, which is the recording parameter of the local microphone. The default value is **AUDIO_SAMPLE_RATE_44100**. Possible values are as follows: AUDIO_SAMPLE_RATE_8000 AUDIO_SAMPLE_RATE_12000 AUDIO_SAMPLE_RATE_11025 AUDIO_SAMPLE_RATE_16000 AUDIO_SAMPLE_RATE_22050 AUDIO_SAMPLE_RATE_24000 AUDIO_SAMPLE_RATE_32000 AUDIO_SAMPLE_RATE_44100 AUDIO_SAMPLE_RATE_48000 AUDIO_SAMPLE_RATE_64000 AUDIO_SAMPLE_RATE_96000
bit_width	No	Integer	Bit width, which is the recording parameter of the local microphone. The default value is **AUDIO_BIT_WIDTH_16**.
nSamples	No	Integer	Number of audio sampling points in each frame, which is the recording parameter of the local microphone. The default value is **1024**, and the value range is [80, 2048].
sound_mode	No	Integer	Audio channel mode, which is the recording parameter of the local microphone. The default value is **AUDIO_SOUND_MODE_MONO**. Possible values are as follows: AUDIO_SOUND_MODE_MONO AUDIO_SOUND_MODE_STEREO

NOTE

- There is only one local microphone. Different recording parameters cannot be set for multiple processes. The parameters set earlier take effect.
- This API and the API in **Playing an Audio File** cannot be used at the same time.

- **Return value**
  - Audio data collector
  - If the construction fails, "CreateError" is reported. You can view skill logs.

5.2 Reading Audio Data

This API is used to read *n* frames of audio data. Only 1.0.8 and later firmware versions are supported.

- **API calling**

  hilens.AudioCapture.read(nFrames)

- **Parameter description**

  Table 5-2 Parameters

Parameter	Mandatory	Type	Description
nFrames	No	Integer	Number of frames that are read. The default value is **1**. A maximum of 512 frames can be read at a time.

- **Return value**

  n frames of audio data. The parameter type is NumPy array (**dtype** is **int16**).

  If the read fails, "RunTimeError" is reported.

5.3 Example - Input

The following is an example of the audio input module:

```
#! /usr/bin/python3.7

import hilens
import wave

def run():
    # Construct a local audio file collector and save the decoded data to a WAV file.
    cap = hilens.AudioCapture("\tmp\test.aac")
    # Construct a local microphone collector.
    cap2 = hilens.AudioCapture(sample_rate=hilens.AUDIO_SAMPLE_RATE_16000,
bit_width=hilens.AUDIO_BIT_WIDTH_16, nSamples=1000,
sound_mode=hilens.AUDIO_SOUND_MODE_MONO)
    wav = wave.open("test.wav", "wb")
    wav.setnchannels(2) # Set the number of channels to 2.
    wav.setsampwidth(2) # Set the sampling rate to 16 bits.
    wav.setframerate(44100) # Set the sampling rate.
    for i in range(100): # Read 500 frames of data and write the data to a file (about 12s).
        data = cap.read(5)
        wav.writeframes(data.tobytes())
    wav.close() # The test.wav audio file is generated in the current directory and can be opened using a
common player.

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


6 Preprocessing Module

6.1 Constructing Image Preprocessors

This API is used to construct and initialize a preprocessor for resize/crop operations (3559 hardware acceleration).

- **API calling**

  hilens.Preprocessor()

- **Return value**

  A preprocessor instance is returned.

  If the construction fails, "CreateError" is reported. You can view skill logs.

6.2 Changing the Image Size

This API is used to change the size of an image.

- **API calling**

  hilens.Preprocessor.resize(src, w, h, t)

- **Parameter description**

Table 6-1 Parameters

Parameter	Mandatory	Type	Description
src	Yes	<class'numpy.ndarray'> object	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.
w	Yes	Positive integer	Width of the image after zoom-in/out. Value range: [64, 1920], a multiple of 2.

Parameter	Mandatory	Type	Description
h	Yes	Positive integer	Height of the image after zoom-in/out. Value range: [64, 1080], a multiple of 2.
t	Yes	Integer 0 or 1	Format of the target image. The value **0** indicates NV21, and value **1** indicates NV12.

- **Return value**

  If the resize operation is successful, the resized image is returned, which is a <class'numpy.ndarray'> object.

  If the resize operation fails, "ValueError" is reported.

6.3 Cropping Images

This API is used to crop an image.

- **API calling**

  hilens.Preprocessor.crop(src, x, y, w, h, t)

- **Parameter description**

  Table 6-2 Parameters

Parameter	Mandatory	Type	Description
src	Yes	<class'numpy.ndarray'> object	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.
x	Yes	Positive integer	X coordinate in the upper left corner of the cropped area. Value range: [0, 1920], a multiple of 2.
y	Yes	Positive integer	Y coordinate in the upper left corner of the cropped area. Value range: [0, 1080], a multiple of 2.
w	Yes	Positive integer	Width of the image after cropping. Value range: [64, 1920], a multiple of 2.
h	Yes	Positive integer	Height of the image after cropping. Value range: [64, 1080], a multiple of 2.
t	Yes	Integer 0 or 1	Format of the target image. The value **0** indicates NV21, and value **1** indicates NV12.

- **Return value**

  If the crop operation is successful, the resized image is returned, which is a <class'numpy.ndarray'> object.

  If the crop operation fails, "ValueError" is reported.

6.4 Converting the Image Color Format

This API is used to convert the image color format. The native OpenCV does not provide the option of converting RGB/BGR to NV12/NV21.

- **API calling**

  hilens.cvt_color(src, code)

- **Parameter description**

  Table 6-3 Parameters

Parameter	Mandatory	Type	Description
src	Yes	<class'numpy.ndarray'> object	Source image (BGR888 or RGB888)
code	Yes	Enumeration type. Options: {RGB2YUV_NV12, RGB2YUV_NV21, BGR2YUV_NV12, BGR2YUV_NV21}.	Conversion type

- **Return value**

  <class'numpy.ndarray'> object, converted image (NV12 or NV21). If the conversion fails, an empty numpy.ndarray object is returned.

6.5 Example - Preprocessing

The following is an example of the preprocessing module:

```
import hilens
import cv2
import numpy as np

def run():
    # Construct a camera.
    cap = hilens.VideoCapture()
    # Obtain a frame of image. The image obtained by the built-in camera is in YUV format.
    # The default resolution of the built-in camera is 720p. Therefore, the YUV image size is (720 x 3/2,1280).
    frame = cap.read()

    # Convert the color format of the image. The YUV-to-BGR conversion needs to be implemented using OpenCV.
    image_bgr = cv2.cvtColor(image_yuv, cv2.COLOR_YUV2BGR_NV21)

     # Convert the color format of the image. The BGR/RGB format can be converted to the YUV format by calling hilens.cvt_color.
    image_yuv = hilens.cvt_color(image_bgr, hilens.BGR2YUV_NV21)
```



```
    # Construct a preprocessor that supports only YUV_NV21/NV12 images.
    proc = hilens.Preprocessor()
     # Adjust the image size.
    resized = proc.resize(image_yuv, 640, 480,0)
    # Crop the image.
    croped = proc.crop(image_yuv, 10, 20, 64, 64, 0)

    # Other processing
    pass

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


7 Model Management Module

7.1 (Optional) Encrypting a Model

HiLens Kit cameras support model encryption. After a model is encrypted, you can call the model using only the HiLens Framework API.

Encrypting a Model

Download the encryption tool **crypto tool**, copy it to the **/tmp** directory of the device system, and grant the execute permission on the tool.

```
chmod +x crypto_tool
```

For details about how to use the tool, see the help information of the tool.

```
./crypto_tool
./crypto_tool encode --model_file plainModelfile --cipher_file cipherModelfile
```

model_file indicates the model file to be encrypted, and **cipher_file** indicates the encrypted model file.

API Calling

The API for an encrypted model is called in the same way as that for an unencrypted model. For details, see:

- **Creating a Model Instance**
- **Performing Inference Using a Model**
- **Example - Model Management**

7.2 Creating a Model Instance

Create a model instance based on the skill model. HiLens Kit cameras can use the models powered by Ascend 310 for inference. Use this method to construct a model for subsequent inference.

When the returned object is destructed, the corresponding model resource is released.

Currently, common models and encrypted models can be created.

- **API calling**

  hilens.Model(filepath)

- **Parameter description**

  Table 7-1 Parameters

Parameter	Mandatory	Type	Description
filepath	Yes	Character string	Path of the model file. You can view the path of the model file by uploading the model file when creating a skill and entering the skill content. • Method 1: Add multiple models to the **Model** field. For details about how to obtain the model path, see **Obtaining Model Paths**. • Method 2: Compress and upload multiple models and code to OBS in advance. Select **Upload from OBS** for **Code Entry Mode**. In this case, the path of the model file is the relative path of the model relative to the file where the code is located.

- **Return value**

  <class'hilens.Model'> model object.

  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).

7.3 Performing Inference Using a Model

After a model is initialized, call the inference API to perform inference using the model. Import a group of data and obtain the inference result. If the type of the input data is not a list consisting of uint8 or float32 arrays, "ValueError" is reported.

- **API calling**

  hilens.Model.infer(inputs)

- **Parameter description**

Table 7-2 Parameters

Parameter	Mandatory	Type	Description
inputs	Yes	List	Inference input. It is a list consisting of a group of uint8 or float32 arrays. Multiple inputs are supported.

- **Return value**

  Model output, which is a list consisting of a group of float arrays. Multiple outputs are supported.

7.4 Example - Model Management

The following is an example of model management:

```
#! /usr/bin/python3.7

import hilens
import numpy as np
def run():

    # Construct a camera.
    cap = hilens.VideoCapture()
    # Obtain a frame of image. The image obtained by the built-in camera is in YUV_NV21 format. The default resolution is 720p.
    frame = cap.read()

    # Load the model.
    # filepath cannot be a file name only. If the model and program are in the same directory, the relative path should be ./my_model.om.
    # If the model is added on the skill development page, use hilens.get_model_dir() to obtain the directory where the model is located. The directory should be as follows:
    # model = hilens.Model(hilens.get_model_dir() + "my_model.om")
    # If there are multiple models, load them separately.
    model1 = hilens.Model("./my_model1.om")
    model2 = hilens.Model("./my_model2.om")
    model3 = hilens.Model("./my_model3.om")

    # Assume that the input of model 1 is a 480 x 480 YUV_NV21 image and the data type is uint8.
    pro = hilens.Preprocessor()
    input1 = pro.resize(frame, 480, 480, 1)
    input1 = input1.flatten()
    # Perform inference.
    output1 = model1.infer([input1])

    # Assume that the input of model 2 is the output (list) of model 1 and the data type is float32.
    input2 = output1
    # Perform inference.
    output2 = model2.infer(input2)

    # Assume that model 3 has multiple inputs and the data type is float32.
    ip_0 = (sample_data[0]).transpose(0, 3, 1, 2).astype(np.float32).flatten()
    ip_1 = (sample_data[1]).transpose(0, 3, 1, 2).astype(np.float32).flatten()
    ip_2 = (sample_data[2]).transpose(0, 3, 1, 2).astype(np.float32).flatten()
    ip_3 = (sample_data[3]).transpose(0, 3, 1, 2).astype(np.float32).flatten()
    ip_4 = (sample_data[4]).transpose(0, 3, 1, 2).astype(np.float32).flatten()
    # Perform inference.
    output3 = model3.infer([ip_0, ip_1, ip_2, ip_3, ip_4])

    # Other processing
```



```
    pass

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```

If the actual inference input is different from the model input, the inference will fail. In this case, the returned 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. The following is an example:

```
>>> input0 = np.zeros((480*480*3), dtype='uint8')
>>> outputs = model.infer([input0])
2019-09-30 18:44:24,075 [ERROR][SFW] Ascend 310: aiModelManager Process failed, please check your
input. Model info:
inputTensorVec[0]: name=data n=1 c=3 h=480 w=480 size=345600
outputTensorVec[0]: name=output_0_reg_reshape_1_0 n=1 c=6750 h=1 w=1 size=27000
your input size:0: 691200;
>>> outputs
17
>>> type(outputs)
<class 'int'>
```


8 Output Module

8.1 Constructing an Output Display

This API is used to construct a display and output videos (image frames) to the Display class.

- **API calling**

  hilens.Display(type, path=None)

- **Parameter description**

Table 8-1 Parameters

Parameter	Mandatory	Type	Description
type	Yes	Enumeration type. Options: **hilens.HDMI**, **hilens.RTMP**, and **hilens.H264_FILE**.	• **hilens.HDMI**: Data is output to the display through the HDMI interface of the device. Currently, only one channel of data can be displayed on the HDMI. • **hilens.RTMP**: Data is output to the RTMP server in real time for users to view. • **hilens.H264_FILE**: Data is output to a file (H.264-encoded raw video streams) for users to view.
path	No	Character string	If the file type is HDMI, ignore this parameter. If the file type is RTMP, set the path to the URL of the RTMP server (**rtmp://***xxxx*). If the file type is H264_FILE, set the path to the output file path (for example, **hilens.get_workspace_path()+"/out.h264"**).

If the file type is H264_FILE, the generated file is only H.264-encoded raw video streams and does not contain information such as the frame rate. In addition, the HiLens Framework does not limit the file size. Therefore, you are advised to use this function only for debugging. If you need to save large files, you are advised to set the file location to **/var/lib/docker**.

- **Return value**

  A display instance is returned.

  If the construction fails, "CreateError" is reported. You can view skill logs or run the **cat /dev/logmpp** command to locate the fault.

8.2 Outputting a Frame of Image

This API is used to display an image. When this API is called for the first time, the Display module sets the video size based on the size of the input images. In subsequent calling operations, the skill must ensure that the size of the input images is the same as that of the original images. The images to be displayed must be in NV21 format. Note that HDMI supports only one output channel and the width and height of the output image must be greater than or equal to 128. Otherwise, the output fails.

- **API calling**

  hilens.Display.show(frame)

- **Parameter description**

  Table 8-2 Parameters

Parameter	Mandatory	Type	Description
frame	Yes	<class 'numpy.ndarray'> type	Image to be displayed. The image must be in NV21 format.

- **Return value**

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

8.3 Uploading Files

This API is used to upload files to OBS. During the file upload, threads are blocked. They are blocked until the file upload is completed. The root directory (location of the target OBS bucket) to which the files are uploaded is configured by the user for each device on the Huawei HiLens console (for details, see **User Guide** > **Managing Data**). If the user does not configure the root directory for a device, the files fail to be uploaded.

- **API calling**

  hilens.upload_file(key, filepath, mode)

  For firmware versions later than 1.0.6, use **hilens.upload_file_to_obs(key, filepath, mode)**.

- **Parameter description**

  Table 8-3 Parameters

Parameter	Mandatory	Type	Description
key	Yes	Character string	Path on OBS to which files are uploaded. Only the OBS file path is needed, for example, **test/output.jpg**, so do not add the website information. Note that names of the two directories in **key** cannot start or end with periods (.) or contain consecutive slashes, such as //.
filepath	Yes	Character string	Absolute path of the file to be uploaded.
mode	Yes	Character string	Upload mode. The options are **write** (overwrite) and **append** (add).

- **Return value**

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

8.4 Uploading Buffer Data

This API is used to upload buffer data to OBS. During the buffer upload, threads are blocked. They are blocked until the upload is completed. The root directory (location of the target OBS bucket) to which the files are uploaded is configured by the user for each device on the Huawei HiLens console (for details, see **User Guide** > **Managing Data**). If the user does not configure the root directory for a device, the files fail to be uploaded.

- **API calling**

  hilens.upload_bufer(key, buffer, mode)

  For firmware versions later than 1.0.6, use **upload_buffer_to_obs(key, buffer, mode)**.

- **Parameter description**

Table 8-4 Parameters

Parameter	Mandatory	Type	Description
key	Yes	Character string	Path on OBS to which files are uploaded. Only the OBS file path is needed, for example, **test/output.txt**, so do not add the website information. Note that names of the two directories in **key** cannot start or end with periods (.) or contain consecutive slashes, such as //.
buffer	Yes	Character string or byte array	Content to be uploaded. The value type can be str or bytes.
mode	Yes	Character string	Upload mode. Two options are available, **write** and **append**. **write** indicates the overwriting mode, and **append** indicates the appending mode.

- **Return value**

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

8.5 Sending Messages

In some scenarios, a skill needs to send messages to users' mobile phones or email addresses. For example, a skill can detect strangers and needs to send messages to the users after detecting strangers. You can call the following API to implement this function:

- **API calling**

  hilens.send_msg(subject, message)

  Only firmware 1.0.7 and later versions provide this API.

- **Parameter description**

Table 8-5 Parameters

Parameter	Mandatory	Type	Description
subject	Yes	Character string, whose length cannot be 0	Subject configured when a message is sent. If the message is sent to the user's mailbox, this field is the subject of the email. If the message is sent to the user's mobile phone, this field is invalid. You can set the message sending mode (by email or mobile phone) when using the skill. To set the message sending mode, log in to the Huawei HiLens console and set the mode. For details, see Subscribing to Messages.
message	Yes	Character string	Message content to be sent.

- **Return value**

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

8.6 Playing an Audio File

This API is used 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**

  hilens.play_aac_file(file_path, vol)

- **Parameter description**

  Table 8-6 Parameters

Parameter	Mandatory	Type	Description
file_path	Yes	Character string	Absolute path of the local audio file
vol	Yes	Integer	Volume of the audio file to be played. The value range is [-121, 6].

- **Return value**

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

8.7 Example - Output

This example shows how to call APIs of multiple output ends. Before using the APIs, ensure that all output ends are connected and available. If an output end does not meet the requirements, comment out or delete the corresponding code in the sample code, and run the sample code. The following is an example of the output module:

```
#! /usr/bin/python3.7
import hilens
import cv2
import numpy as np
import wave

def run():
    # Show the display connected to the HDMI.
    # Currently, only one channel of data can be displayed on the HDMI. If multiple skills are displayed on the HDMI at the same time, an error is reported.
    disp0 = hilens.Display(hilens.HDMI)
    # Push streams to the server whose address is rtmp://192.168.1.1/stream.
    disp1 = hilens.Display(hilens.RTMP, "rtmp://192.168.1.1/stream")
    # Write the video to a file. The file generated by hilens.H264_FILE is the raw video stream file encoded using H.264.
    # The file size is not limited. It is recommended that this command be used only for debugging.
    disp2 = hilens.Display(hilens.H264_FILE, hilens.get_workspace_path() + "video.h264")
    # hilens.get_workspace_path() returns the skill workspace directory. For details, see the resource management module.

    # Construct a local camera video collector.
    cap = hilens.VideoCapture()

    # Display the video to the HDMI display device.
    disp0.show(cap.read())

    # Upload video.h264 to OBS.
    # Generate a video file in H.264 format.
    disp2.show(cap.read())
    # Upload the file to OBS.
    hilens.upload_file_to_obs("video", hilens. get_workspace_path() + "video.h264", "write")

    # Append 1234 to the test4 file of OBS.
    hilens.upload_buffer_to_obs("test4", "1234", "append")

    # Upload images to OBS using the data in the buffer.
    # Convert the images to the BGR format.
    frame = cap.read()
    img_bgr = cv2.cvtColor(frame, cv2.COLOR_YUV2BGR_NV21)
    # Encode the current images in JPG format.
    img_encode = cv2.imencode(".jpg", img_bgr)[1]
    # Use upload_bufer to upload the images in the buffer. The image format must be the same as the encoding format.
    hilens.upload_bufer("img.jpg", img_encode, "write")

    # Play an audio file.
    hilens.play_aac_file("test.aac", 6)

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


9 EIServices Module

9.1 Introduction

The EIServices module provides convenient APIs for you to quickly call various AI services on HUAWEI CLOUD. For details about the AI services, see the related service documentation.

Currently, two types of APIs are provided: common APIs and encapsulated APIs. These APIs are applicable only to firmware 1.0.7 and later versions that call APIs in the **CN North-Beijing4** region.

9.2 Common APIs

Common APIs can be used to access various AI services on HUAWEI CLOUD. For details about the AI services, see the related service documentation.

- **API calling**

  hilens.EIServices.Request(method, host, uri, queryParams, payload, headers)

- **Parameter description**

  Table 9-1 Parameters

Parameter	Mandatory	Type	Description
method	Yes	Enumeration	Request method. The value can be **hilens.GET**, **hilens.POST**, **hilens.PUT**, or **hilens.DELETE**.
host	Yes	Character string	Request domain name. **host** and **uri** comprise a complete request URL.
uri	Yes	Character string	Request URI. **host** and **uri** comprise a complete request URL.

Parameter	Mandatory	Type	Description
queryParams	Yes	Character string	String to be queried.
payload	Yes	Character string	Request body.
headers	Yes	Character string	Request header. The **hilens.EIHeaders()** object is provided for adding request headers. The following is an example: headers = hilens.EIHeaders() headers.add("Content-Type: application/json")

- **Return value**

  EIResponse structure, including **requestState** and **responseBody**. For details, see **Table 9-2**.

  Table 9-2 Return value description

Parameter	Type	Description
EIResponse.requestState	Boolean	Request status. **True** indicates a success, and **False** indicates a failure.
EIResponse.responseBody	Character string	Request response body.

9.3 Encapsulated APIs

Encapsulated APIs are used for human figure and face detection, facial attribute recognition, license plate recognition, dog excretion detection, and Face Recognition in HUAWEI CLOUD EI services. These APIs are encapsulated to facilitate use. Contact Huawei HiLens personnel to enable the APIs for human figure and face detection, facial attribute recognition, license plate recognition, and dog excretion detection. Enable the API for the Face Recognition service on the Face Recognition console.

- **API calling**

  Human figure and face detection:

  hilens.EIServices.HumanDetect(image_base64)

  Facial attribute recognition:

  hilens.EIServices.FaceAttribute(image_base64)

License plate recognition:

hilens.EIServices.LicensePlate(image_base64)

Dog excretion detection:

hilens.EIServices.DogShitDetect(image_base64)

Face retrieval:

hilens.EIServices.SearchFace(face_set_name, image_base64, top_n, threshold, filter)

Adding a face to the face library:

hilens.EIServices.AddFace(face_set_name, image_base64, external_image_id)

Face detection:

hilens.EIServices.FaceDetect(image_base64, attributes)

Face verification:

hilens.EIServices.FaceCompare(image1_base64, image2_base64)

- **Parameter description**

  Table 9-3 Parameters

Parameter	Mandatory	Type	Description
image_base64, image1_base64, image1_base64	Yes	Character string	Base64 code of the image to be recognized.
face_set_name, top_n, threshold, filter	Yes	Character string	Face search parameters.
face_set_name, external_image_id	Yes	Character string	Face adding parameters.
attributes	Yes	Character string	Facial attributes. This parameter indicates whether to return the facial attribute list. Multiple attributes are separated by commas (,).

- **Return value**

  EIResponse structure, including **requestState** and **responseBody**. For details, see **Table 9-4**.

Table 9-4 Return value description

Parameter	Type	Description
EIResponse.requestState	Boolean	Request status. **True** indicates a success, and **False** indicates a failure.
EIResponse.responseBody	Character string	Request response body.

9.4 Example - EIServices Module

The following is an example of the output code of the EIServices module:

```
import hilens
import cv2
import numpy as np
import base64
import json

def run():
    # Use an image as the input.
    f=open('/tmp/dengchao.jpg','rb')
    base_f=base64.b64encode(f.read())
    f_string=base_f.decode('utf-8')
    response0 = hilens.EIServices.HumanDetect(f_string)
    print(response0.requestState)
    print(response0.responseBody)

    # Use the Mat format or directly input the image from the camera.
    #img = cv2.imread("/tmp/dengchao.jpg")
    cap = hilens.VideoCapture()
    frame = cap.read()
    img = cv2.cvtColor(frame, cv2.COLOR_YUV2BGR_NV21)
    img_str = cv2.imencode('.jpg', img) [1].tostring() # Encode the image into stream data, store the stream data in the memory buffer, and convert the stream data into the string format.
    b64_code = base64.b64encode(img_str) # Encode the image into the Base64 format.
    f_string1=b64_code.decode('utf-8')
    response1 = hilens.EIServices.HumanDetect(f_string1)
    print(response1.requestState)
    print(response1.responseBody)

    headers = hilens.EIHeaders()
    body = {"image_base64": f_string}
    json_str = json.dumps(body)
    response5 = hilens.EIServices.Request(hilens.POST, "hilens-api.cn-north-4.myhuaweicloud.com", "/v1/human-detect", "", json_str, headers)
    print(response5.requestState)
    print(response5.responseBody)
    body1 = {"face_set_name": "ei_test"}
    json_str1 = json.dumps(body1)
    response6 = hilens.EIServices.Request(hilens.POST, "face.cn-north-4.myhuaweicloud.com", "/v1/fc3bc995e9c441369d71159c67404e88/face-sets", "", json_str1, headers)
    print(response6.requestState)
    print(response6.responseBody)
    response7 = hilens.EIServices.AddFace("ei_test", f_string, "")
    print(response7.requestState)
    print(response7.responseBody)
    response8 = hilens.EIServices.SearchFace("ei_test", f_string, 1, 0.93, "")
    print(response8.requestState)
    print(response8.responseBody)
```



```
    response9 = hilens.EIServices.Request(hilens.POST, "face.cn-north-4.myhuaweicloud.com","/v1/
fc3bc995e9c441369d71159c67404e88/face-sets/ei_test/search", "", json_str, headers)
    print(response9.requestState)
    print(response9.responseBody)
    response10 = hilens.EIServices.Request(hilens.DELETE, "face.cn-north-4.myhuaweicloud.com", "/v1/
fc3bc995e9c441369d71159c67404e88/face-sets/ei_test", "", "", headers)
    print(response10.requestState)
    print(response10.responseBody)
    response11 = hilens.EIServices.Request(hilens.GET, "face.cn-north-4.myhuaweicloud.com", "/v1/
fc3bc995e9c441369d71159c67404e88/face-sets/ei_test", "", "", headers)
    print(response11.requestState)
    print(response11.responseBody)

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


10 Resource Management Module

10.1 Obtaining Model Paths

This API is used to obtain the path of the directory where the skill model is located. The path ends with a slash (/). This function is applicable to the scenario where a model is selected and delivered on the model management page during skill creation. The model file is downloaded to a dedicated location for storing models. This function is used to obtain the directory where the model is located. If the HiLens Framework does not obtain the directory where the model is located, the current path (the directory where the code is stored) is returned.

- **API calling**

  hilens.get_model_dir()

- **Return value**

  A character string is returned, indicating the path of the skill model. If the operation fails, an empty character string is returned.

10.2 Obtaining the Directories of the Skill Workspaces

This API is used to obtain the path of the directory where the skill workspace is located. The path ends with a slash (/). You are not advised to write data in the skill installation directory. Therefore, you need to specify the writable workspace for each skill. If the HiLens Framework does not obtain the workspace location, the current path is returned.

- **API calling**

  hilens.get_workspace_path()

- **Return value**

  A character string is returned. The absolute path of the working directory is **/.../data/**. If the operation fails, an empty character string is returned.

10.3 Obtaining Skill Configurations

This API is used to obtain skill configurations. If the information fails to be obtained, **None** is returned.

- **API calling**

  hilens.get_skill_config()

- **Return value**

  If the operation is successful, a skill configuration dict is returned. If the operation fails, **None** is returned.

10.4 Downloading a File from OBS

This API is used to download a file from OBS.

- **API calling**

  hilens.download_from_obs(url, download_to)

- **Parameter description**

  Table 10-1 Parameters

Parameter	Mandatory	Type	Description
url	Yes	Character string	OBS resource link. For details on how to obtain the resource link, see **Accessing an Object Using Its URL** in the *Object Storage Service Console Operation Guide*.
download_to	Yes	Character string	Directory where the downloaded file is stored. You are advised to use an existing directory. The path contains a maximum of 256 characters.

- **Return value**

  The value **0** indicates a success, and other values indicate a failure.

10.5 Calculating the MD5 Values of Files

This API is used to calculate the md5 values of files.

- **API calling**

  hilens.md5_of_file(file)

- **Parameter description**

Table 10-2 Parameters

Parameter	Mandatory	Type	Description
file	Yes	Character string	Path of the file to be calculated.

- **Return value**

  The md5 values of files are returned.

10.6 Example - Resource Management

The following is an example of resource management:

```
#! /usr/bin/python3.7

import hilens
import os

def run():
    # Obtain the path of the directory where the skill workspace is located. The path ends with a slash (/).
    skill_path = hilens.get_workspace_path()

    # Obtain the path of the directory where the skill model is located. The path ends with a slash (/).
    model_path = hilens.get_model_dir()

    # Obtain the skill configuration. If the information fails to be obtained, None is returned.
    skill_config = hilens.get_skill_config()
    # Assume that the face_dataset configuration item exists in the skill configuration and its value is the address of the face library file face_dataset.zip in OBS.
# Set skill configuration parameters. For details about how to set skill configuration parameters, see the User Guide.
    face_dataset_url = skill_config["face_dataset"]["value"]
    # Download the file from OBS to the skill workspace directory and check whether the download is successful based on the return value.
    ret = hilens.download_from_obs(face_dataset_url, hilens.get_workspace_path())
    if ret != 0:
        hilens.error("Failed to download from obs")
        return
    # Create a folder in the skill workspace directory and decompress it.
    os.system('mkdir '+hilens.get_workspace_path()+'face_dataset')
    os.system('unzip '+hilens.get_workspace_path()+'face_dataset.zip'+' -d '+hilens.get_workspace_path()+'face_dataset/')
        # Calculate the MD5 values of files.
    md5 = hilens.md5_of_file(hilens.get_workspace_path()+"face_dataset.zip")

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


NOTE

To set the skill configuration parameters, do as follows:

1. Log in to the Huawei HiLens console and set runtime parameters during skill development. For details, see **Creating Skills** in the *User Guide*.
2. After the skill development is complete, deploy the skill on your device. For details, see **Deploying and Debugging Skills** in the *User Guide*.
3. After deploying the skill to your device, you can set runtime configuration parameters on the **Skill Management** page. For details, see **Adding Runtime Configurations** in the *User Guide*.

11 Hard Example Upload Module

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 Framework Firmware Version**.

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 Initializing

This API is used to construct a hard example filter.

- **API calling**

  hilens.HardSample(threshold_one, threshold_two, filter_type)

- **Parameter description**

  Table 11-1 Parameters

Parameter	Mandatory	Type	Description
threshold_one	Yes	float	Threshold. The value varies depending on the value of **filter_type**.
threshold_two	Yes	float	Threshold. The value varies depending on the value of **filter_type**.

Parameter	Mandatory	Type	Description
filter_type	Yes	int	Type of a hard example filter. Possible values are **0**, **1**, and **2**, which correspond to **CrossEntropyFilter**, **IBT**, and **CSF**, respectively. For details, see **Introduction to Hard Example Upload**. • If the value is **0**, **threshold_one** corresponds to **threshold_cross_entropy** of **CrossEntropyFilter**, and **threshold_two** can be any value. • If the value is **1**, **threshold_one** and **threshold_two** correspond to **box_threshold** and **img_threshold** of **IBT**, respectively. • If the value is **2**, **threshold_one** and **threshold_two** correspond to **box_threshold_low** and **box_threshold_up** of **CSF**, respectively.

- **Return value**
  A hard example filter. If the construction fails, an exception is thrown.

11.3 Determining a Hard Example Image in a Detection Algorithm

This API is used to check the detection result.

- **API calling**
  hard_sample_detection_filter(inputs)
- **Parameter description**

Table 11-2 Parameters

Parameter	Mandatory	Type	Description
inputs	Yes	list	Detection box, for example, **[bbox1, bbox2, bbox3,....]**. In **bbox = [xmin, ymin, xmax, ymax, score, label]**, **xmin**, **ymin**, **xmax**, **ymax**, and **score** are of the float type, the value range of **score** is [0,1], and **label** is of the int type.

- **Return value**

  A Bool value is returned. **True** indicates that the image is a hard example, and **False** indicates that the image is not a hard example.

11.4 Determining a Hard Example Image in a Classification Algorithm

This API is used to determine the classification result.

- **API calling**

  hard_sample_classification_filter(inputs,input_size)

- **Parameter description**

Table 11-3 Parameters

Parameter	Mandatory	Type	Description
inputs	Yes	list	Class score, for example, **[class1-score, class2-score, class2-score,....]**, where **class-score** ranges from 0 to 1
input_size	Yes	int	Number of classes

- **Return value**

  A Bool value is returned. **True** indicates that the image is a hard example, and **False** indicates that the image is not a hard example.

11.5 Uploading a Hard Example Image Set

This API is used to upload a hard example image set. The name of the uploaded file is in the format of **model_name-camera_name-index.jpg**. If a file with the same name exists in the folder, the file will be overwritten. You can add a timestamp to **model_name** or **camera_name** to prevent the file from being

overwritten. Note that the OBS file name can contain a maximum of 1,024 characters.

- **API calling**

  upload_jpeg(upload_url, index, model_name, camera_name, frame)

- **Parameter description**

  Table 11-4 Parameters

Parameter	Mandatory	Type	Description
upload_url	Yes	string	URL of the hard example set to be uploaded. To obtain the URL of a hard example set, configure the corresponding dataset on the hard example upload page and then use **get_hard_sample_config** to obtain the hard example configurations. For details, see **Obtaining Hard Example Configurations**. The **dataset_path** parameter in the return value corresponds to the URL of the dataset.
index	Yes	int	Sequence number of the image to be uploaded
model_name	Yes	string	Name of the model corresponding to the image to be uploaded
camera_name	Yes	string	Name of the camera corresponding to the image to be uploaded
frame	Yes	mat	Image to be uploaded. The image must be in NV21 format.

- **Return value**

  A Bool value is returned. **True** indicates that the image is successfully uploaded, and **False** indicates that the image fails to be uploaded.

11.6 Obtaining Hard Example Configurations

This API is used to read the hard example configuration file. The path of the hard example configuration file is the **data** directory of the corresponding skill, for example, **/home/hilens/skills/***skill_path/data/hardsample_config.json**.

- **API calling**

  hilens.get_hard_sample_config()

- **Return value**

  Hard example configuration file in JSON format. Example:

```
{
"hard_sample_setting" :
   [
     {
       "camera_names" :["123"],
       "data_count" : 100,
       "datacur_count" : 100,
       "dataset_name" : "dataset-a3ae",
       "dataset_path" : "https://a.b.csss.obs.cn-north-7.ulanqab.huawei.com/nali/",
       "model_algorithm" : "image_classification",
       "model_id" : "073c4c8674164307ae300b713a4a050c",
       "model_name" : "model-framework5",
       "setting_config" :
          {
             "thr" : 0.5
          }
     }
   ]
}
```

Table 11-5 Parameters

Parameter	Description
camera_names	Camera name. Data of different cameras can be distinguished during data upload.
data_count	Total number of uploaded images
datacur_count	Number of images that have been uploaded
dataset_name	Name of the target dataset
dataset_path	Target path, which corresponds to the URL of the dataset
model_algorithm	Algorithm of a model. Generally, the algorithm is classification or detection.
model_id	Model ID
model_name	Model name. Data of different models can be distinguished during data upload.
setting_config	Other settings. Some configurations are custom, such as the threshold.
thr	Threshold. You can deliver thresholds on the hard example upload configuration page so that different thresholds can be used in different scenarios.

11.7 Updating Hard Example Configurations

This API is used to update hard example configurations to the hard example configuration file, and update the hard example upload status on the cloud based on the input.

- **API calling**

  hilens.set_hard_sample_config(conf)

- **Parameter description**

  Table 11-6 Parameters

Parameter	Mandatory	Type	Description
conf	Yes	json	Hard example configuration to be updated

- **Return value**

  HiLensEC error code. If the operation is successful, **0** is returned. Otherwise, the operation fails. For details, see **Error Codes**.

11.8 Example - Hard Example Upload

The following is an example of uploading a hard example:

```
import hilens
import cv2
import numpy as np

def run():
    # Construct a camera.
    cap = hilens.VideoCapture()

    disp = hilens.Display(hilens.HDMI)

    hard_sample = hilens.HardSample(0.5,0.5,1) # 1 and 2 is the algorithm used by the detection model.
     #hard_sample = hilens.HardSample(0.5,0.5,0) # 0 is the algorithm used by the classification model.

    hard_sample_flag = False # Whether hard sample upload configuration exists

    hard_sample_config = hilens.get_hard_sample_config() # Obtain hard example configurations.
    if not hard_sample_config:
        hilens.warning("hardSampleConfig is empty")
    else:
        hard_sample_flag = True
        data_count = hard_sample_config["hard_sample_setting"][0]["data_count"]
        data_current_count = hard_sample_config["hard_sample_setting"][0]["datacur_count"]
        upload_jpeg_url = hard_sample_config["hard_sample_setting"][0]["dataset_path"]
        model_name = hard_sample_config["hard_sample_setting"][0]["model_name"]
        camera_name = "default"
        if data_count > data_current_count:
            upload_flag = True # Whether to continue the upload
        else:
            upload_flag = False

    while True:
```



```
        # Obtain a frame of image.
        frame = cap.read()
        if hard_sample_flag:
            if upload_flag:
                if hard_sample.hard_sample_detection_filter([[0.,0.,1280.,720.,0.4,1]]): # The input of the
detection algorithm is the post-processed bounding box. Each bounding box contains [xmin, ymin, xmax,
ymax, conf, label] (including the confidence score and class labels).
                #if hard_sample.hard_sample_classification_filter([0.2, 0.2, 0.2, 0.2, 0.2],5):  # The input of the
classification algorithm is the probability of each class, that is, the output of the model.
                    hard_sample.upload_jpeg(upload_jpeg_url, data_current_count, model_name, camera_name,
frame)
                    data_current_count += 1
                    hard_sample_config["hard_sample_setting"][0]["datacur_count"] = data_current_count
                    if data_current_count == 1 or data_current_count == data_count:
                        if data_current_count == data_count:
                            upload_flag = False
                    hilens.set_hard_sample_config(hard_sample_config) # Update the hard example configuration
file on the device.

        #Output the result to HDMI.
        disp.show(frame)

if __name__ == '__main__':
    #The value of hello must be the same as that of the check value in the basic information. For details,
see the developer guide.
    hilens.init("hello")
    run()
    hilens.terminate()
```


12 Log Module

12.1 Setting Log Levels

This API is used to set log levels. By default, only logs of the Info level or higher are printed. Log levels are: Trace > Debug > Info > Warning > Error > Fatal.

- **API calling**

  hilens.set_log_level(level)

- **Parameter description**

  Table 12-1 Parameters

Parameter	Mandatory	Type	Description
level	Yes	Enumeration	**hilens.TRACE**: logs of the Trace level or higher **hilens.DEBUG**: logs of the Debug level or higher **hilens.INFO**: logs of the Info level or higher **hilens.WARNING**: logs of the Warning level or higher **hilens.ERROR**: logs of the Error level or higher **hilens.FATAL**: logs of the Fatal level

- **Return value**

  None

12.2 Trace Logs

This API is used to output logs of the Trace level and save them to a log file.

- **API calling**

  hilens.trace(msg)

- **Parameter description**

  Table 12-2 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Trace log information. A log record can contain a maximum of 255 characters.

- **Return value**
  None

12.3 Debug Logs

This API is used to output logs of the Debug level and save them to a log file.

- **API calling**
  hilens.debug(msg)
- **Parameter description**

  Table 12-3 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Debug log information. A log record can contain a maximum of 255 characters.

- **Return Value**
  None

12.4 Info Logs

This API is used to output logs of the Info level and save them to a log file.

- **API calling**
  hilens.info(msg)
- **Parameter description**

Table 12-4 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Info log information. A log record can contain a maximum of 255 characters.

- **Return value**
  None

12.5 Warning Logs

This API is used to output logs of the Warning level and save them to a log file.

- **API calling**
  hilens.warning(msg)
- **Parameter description**

Table 12-5 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Warning log information. A log record can contain a maximum of 255 characters.

- **Return value**
  None

12.6 Error Logs

This API is used to output logs of the Error level and save them to a log file.

- **API calling**
  hilens.error(msg)
- **Parameter description**

Table 12-6 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Error log information. A log record can contain a maximum of 255 characters.

- **Return value**
  None

12.7 Fatal Logs

This API is used to output logs of the Fatal level and save them to a log file.

- **API calling**
  hilens.fatal(msg)
- **Parameter description**

  Table 12-7 Parameters

Parameter	Mandatory	Type	Description
msg	Yes	Character string	Fatal log information. A log record can contain a maximum of 255 characters.

- **Return value**
  None

12.8 Example - Logs

Log examples are as follows:

```
#! /usr/bin/python3.7

import hilens

def run():
    # Set log levels.
    hilens.set_log_level(hilens.DEBUG)

    # Print a log of the Trace level.
    hilens.trace("trace")

    # Print a log of the Debug level.
    hilens.debug("debug")

    # Print a log of the Info level.
    hilens.info("info")

    # Print a log of the Warning level.
    hilens.warning("warning")

    # Print a log of the Error level.
    hilens.error("error")

    # Print a log of the Fatal level.
    hilens.fatal("fatal")

if __name__ == '__main__':
    hilens.init("hello")
    run()
    hilens.terminate()
```


13 Error Codes

The HiLens Framework returns error codes using enumeration types. If an error occurs when an API is called and an error code is returned, you can view the following enumeration types to obtain the error information:

Table 13-1 Error codes

Error Code	Description
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 permission to access the file.
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 timeout.
AUTH_FAILED	Incorrect authentication information. Check whether the AK, SK, and token are valid.
NOT_FOUND	The object does not exist.
SERVER_ERROR	Internal service error.

Error Code	Description
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 constraints.
INVALID_DST_SIZE	During image processing, the dst size does not meet the constraints.
MPP_PROCESS_FAILED	The MPP fails to process images.
WEBSOCKET_ERROR	WebSocket error.
CONFIG_FILE_ERROR	Incorrect configuration file.
INVALID_PARAM	Incorrect parameter.

14 Change History

Date	Description
2019-07-05	Edited the information according to the Skill Framework 0.2.6 version.
2019-02-22	**Modified** • Modified the usage of **upload_buffer** in the data output operations.
2019-01-29	This issue is the first official release.

