API Development Manual:

AMTMultiBio SDK For Android

API Version: V2.1

Doc Version: V1.0

June 2022

Thank you for choosing our product. Please read the instructions carefully before operation. Follow these instructions to ensure that the product is functioning properly. The images shown in this manual are for illustrative purposes only.

For further details, please visit our Company's website www.armatura.us.

Trademark

ARMATURA is a registered trademark of ARMATURA LLC. Other trademarks involved in this manual are owned by their respective owners.

Disclaimer

This manual contains information on the operation and maintenance of the ARMATURA product. The copyright in all the documents, drawings, etc. in relation to the ARMATURA supplied product vests in and is the property of ARMATURA. The contents hereof should not be used or shared by the receiver with any third party without express written permission of ARMATURA.

The contents of this manual must be read as a whole before starting the operation and maintenance of the supplied product. If any of the content(s) of the manual seems unclear or incomplete, please contact ARMATURA before starting the operation and maintenance of the said product.

It is an essential pre-requisite for the satisfactory operation and maintenance that the operating and maintenance personnel are fully familiar with the design and that the said personnel have received thorough training in operating and maintaining the machine/unit/product. It is further essential for the safe operation of the machine/unit/product that personnel have read, understood, and followed the safety instructions contained in the manual.

In case of any conflict between terms and conditions of this manual and the contract specifications, drawings, instruction sheets or any other contract-related documents, the contract conditions/documents shall prevail. The contract specific conditions/documents shall apply in priority.

ARMATURA offers no warranty, guarantee, or representation regarding the completeness of any information contained in this manual or any of the amendments made thereto. ARMATURA does not extend the warranty of any kind, including, without limitation, any warranty of design, merchantability, or fitness for a particular purpose.

ARMATURA does not assume responsibility for any errors or omissions in the information or documents which are referenced by or linked to this manual. The entire risk as to the results and performance obtained from using the information is assumed by the user.

ARMATURA in no event shall be liable to the user or any third party for any incidental, consequential, indirect, special, or exemplary damages, including, without limitation, loss of business, loss of profits, business interruption, loss of business information or any pecuniary loss, arising out of, in connection with, or relating to the use of the information contained in or referenced by this manual, even if ARMATURA has been advised of the possibility of such damages.

This manual and the information contained therein may include technical, other inaccuracies, or typographical errors. ARMATURA periodically changes the information herein which will be incorporated into new additions/amendments to the manual. ARMATURA reserves the right to add, delete, amend, or modify the information contained in the manual from time to time in the form of circulars, letters, notes, etc. for better operation and safety of the machine/unit/product. The said additions or amendments are meant for improvement /better operations of the machine/unit/product and such amendments shall not give any right to claim any compensation or damages under any circumstances.

ARMATURA shall in no way be responsible (i) in case the machine/unit/product malfunctions due to any non-compliance of the instructions contained in this manual (ii) in case of operation of the machine/unit/product beyond the rate limits (iii) in case of operation of the machine and product in conditions different from the prescribed conditions of the manual.

The product will be updated from time to time without prior notice. The latest operation procedures and relevant documents are available on http://www.armatura.us.

If there is any issue related to the product, please contact us.

ARMATURA Headquarters

Address 190 Bluegrass Valley Pkwy,
Alpharetta, GA 30005, USA.

For business-related queries, please write to us at: info@armatura.us.

To know more about our global branches, visit www.armatura.us.

About the Company

ARMATURA is a leading global developer and supplier of biometric solutions which incorporate the latest advancements in biometric hardware design, algorithm research & software development. ARMATURA holds numerous patents in the field of biometric recognition technologies. Its products are primarily used in business applications which require highly secure, accurate and fast user identification.

ARMATURA biometric hardware and software are incorporated into the product designs of some of the world's leading suppliers of workforce management (WFM) terminals, Point-of-Sale (PoS) terminals, intercoms, electronic safes, metal key lockers, dangerous machinery, and many other products which heavily rely on correctly verifying & authenticating user's identity.

About the Manual

This manual introduces the operations of **AMTMultiBio SDK For Android**.

All figures displayed are for illustration purposes only. Figures in this manual may not be exactly consistent with the actual products.

Document Conventions

Conventions used in this manual are listed below:

GUI Conventions

For Software	
Convention	**Description**
Bold font	Used to identify software interface names e.g., **OK**, **Confirm**, **Cancel**
>	Multi-level menus are separated by these brackets. For example, File > Create > Folder.
For Device	
Convention	**Description**
< >	Button or key names for devices. For example, press <OK>
[ ]	Window names, menu items, data table, and field names are inside square brackets. For example, pop up the [New User] window
/	Multi-level menus are separated by forwarding slashes. For example, [File/Create/Folder].

Symbols

Convention	Description
	This implies about the notice or pays attention to, in the manual
	The general information which helps in performing the operations faster
	The information which is significant
	Care taken to avoid danger or mistakes
	The statement or event that warns of something or that serves as a cautionary example.

Table of Contents

1 Introduction

This document will provide the basic development guide and technical background to help with the better implementation of AMTMultiBio SDK for Android. From the perspective of a developer, the key design objective of this SDK is its compatibility and ease of execution.

This development manual contains the product development documentation for developers that describes the functions provided by the SDK and its related usage which eases the development environment. The following sections will explain all the required information on how to perform and integrate the AMTMultiBio SDK.

1.1 Overview of the SDK

AMTMultiBio SDK is a developer-friendly software development kit for application integration with Armatura face and palm multimodal biometric modules. It provides the UVC (USB Video Class) and HID (Human Interface Devices) communication interfaces to the hardware modules and many biometric interfaces to face and palm recognition process.

The SDK not only encapsulates the module-built-in face and palm recognition algorithm functions, also provides the low computing power, high-performance on-host face/palm matching methods. This provides the flexibility to the application integration and simplify the solution by managing and matching templates directly in the application.

AMTMultiBio SDK supports common operating system including Windows, Android, and Linux (on request), empowers the biometric features on wide range of hardware/software applications, especially the low computing powered embedded hardware applications.

1.2 Feature of the SDK

- **High-performance and High-accuracy Face and Palm Recognition**

  With the cutting-edge, deep learning-based computer vision technologies, the high-accurate recognition of face and palm can be completed within a second.

- **Rich Communication Interfaces with Module**

  Support UVC and HID protocols, UVC is used for real-time video streaming communication, while HID is used for data communication such as configuring the hardware modules, syncing the user data, retrieving the generated templates and matching result from the hardware module, and more.

- **Generate Template on Module**

  For deep learning technologies, heavy computing power is required to extract biometric

features and generate face/palm templates. With MultiBio SDK, the template generation task is shifted to Armatura multimodal modules, which greatly lowers the hardware platform requirements and entitles low computing powered devices (i.e. MCU-based) to biometric technologies.

- **Multiple Matching Modes: Match on Host and Match on Module**

  Match on Host is provided to the application running on powerful CPU platform, the application can run the face/palm template matching directly without syncing the user/templates with the devices. This simplifies the integration solution.

  While on special cases where the host computing power is weak or has limited storage space, Match on Module provides the suitable solution without modifying the hardware.

- **Face Attribute Analysis**

  The face recognition interfaces provide high-accurate face attribute analysis functions which can estimate the age, gender, emotion classification, and detect beard, glasses, hat, and face mask. Such functions are very suitable for public business application where anonymity is required.

- **High-Accurate Liveness Detection**

  With deep learning algorithm and anti-spoofing capability based on near-infrared light images collected by the multimodal modules, MultiBio SDK can effectively block attacks from digital photos, printed color, black and white photos, and videos.

- **High-Tolerance to Face and Palm Postures**

  The module built-in algorithms interfaced by MultiBio SDK not only tolerates large angle range of Pitch, Yaw and Roll postures of the face and palm, but also effectively identifies various palm shapes from tensed to bended. The high posture tolerance allows user to perform face or palm recognition in a natural way, which greatly improves the user experience.

- **High-Adaptability to Various Environments**

  Using regional-image enhancement technology, the area of the detected face or palm is enhanced to create high quality image for recognition process. This method effectively reduces interference from ambient light, making it highly adaptable to various changing lighting conditions.

- **Support for Multiple Operating Systems**

  The standard MultiBio SDK supports Windows and Android operating systems, upon customer request, we can customize the SDK and fit to multiple Linux variants from PC version to embedded version.

1.3 Advantage of the SDK

- Easy to use by other developers.
- Thorough documentation to explain how your code works.
- Enough functionality so it adds value to other applications.

- Does not negatively impact.
- Plays well with other SDKs.

2 Technical Specifications

This SDK provides an ARR package that supports Java language development. It must be used on Android 4.1 or later, which supports the Android USB host.

2.1 SDK Architecture

AMTMultiBio SDK is suitable for various types of devices, for specific performances, and support multiple platforms. It provides two matching modes, one is Match-on-Host, and the other is Match-on-Module.

The Match-on-Host mode can transmit images and video data of face and palm through UVC at high speed. Also, it can use the advantages of the high-performance host, which quickly realizes face and palm recognition.

The Match-on-Module mode utilizes the performance and capacity of the module to achieve stable and effective data interaction through the HID protocol. And this reduces the burden on the Host side with low performance, small capacity and not supporting the UVC protocol.

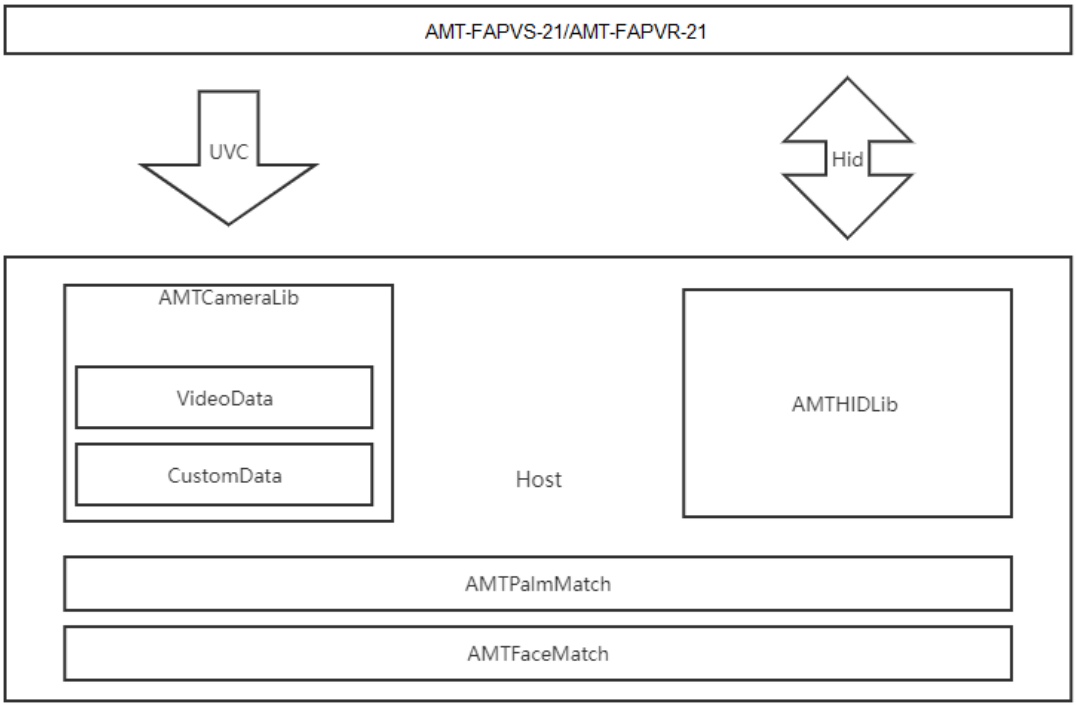
The flexible use of different modes enables rapid comparison of large-capacity faces and palms and rapid application development.

2.1.1 Match-on-Host Mode

If the user uses the Match-on-Host mode, the SDK architecture is mainly composed of **AMTCameraManager**, **AMTHIDManager**, **AMTFaceMatch**, and **AMTPalmMatch**.

The UVC collector is realized by **AMTCameraManager**, the HID communication is realized by **AMTHidManager**, the face Match-on-Host is accomplished by **AMTFaceMatch**, and the palm Match-on-Host comparison is performed by **AMTPalmMatch**.

The overall structure of the SDK is shown in the following figure.



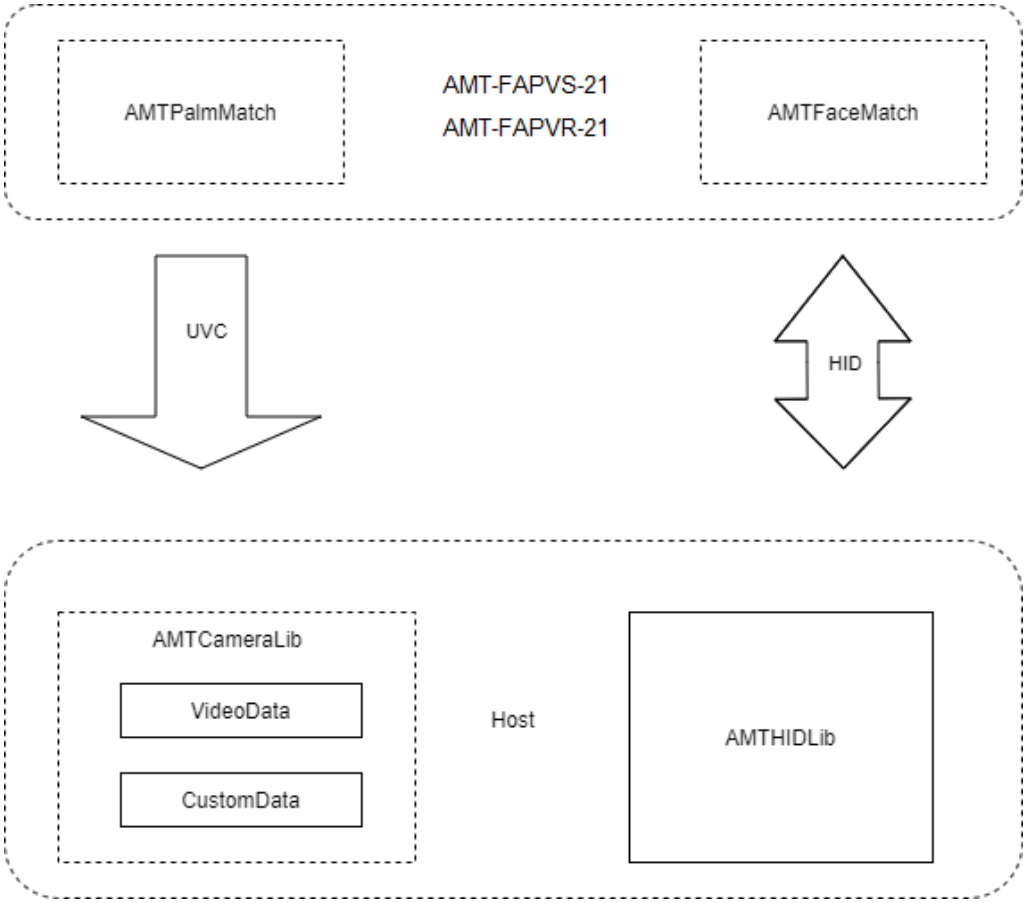
Process Description

- **AMTCameraManager** acquires the image data (MJPEG format) and custom data (including biometric information such as faces, palms, etc.).
- **AMTHidManager** can setup modules (light control, AE, etc.), biometric algorithms (face angle, face size, verification result, etc.) and manage the module data (user, template, matching record).
- For host-side palm recognition, **AMTPalmMatch** is used.
- For host-side face recognition **AMTFaceMatch** is used.

2.1.2 Match-on-Module Mode

If the user uses the Match-on-Module mode, the SDK architecture is mainly composed of AMTCameraManager and AMTHIDManager. The UVC collector is realized by [AMTCameraManager], the HID communication is realized by [AMTHidManager].

The below image describes the overall structure of the SDK:



- The module-side verification not only transmits face and palm data through UVC but also obtains the user's face and palm data through [Polling Recognition Result].
- The host side does not need to use AMTPalmMatch and AMTFaceMatch for comparison processing and can directly obtain match information through UVC or [Polling Recognition Result].

2.2 Application Scenario

By providing a complete and rich SDK, developers can call and configure the corresponding functional interface according to their requirements and develop their applications. And also integrate face or palm recognition functions on the client's host side, which considerably shortens

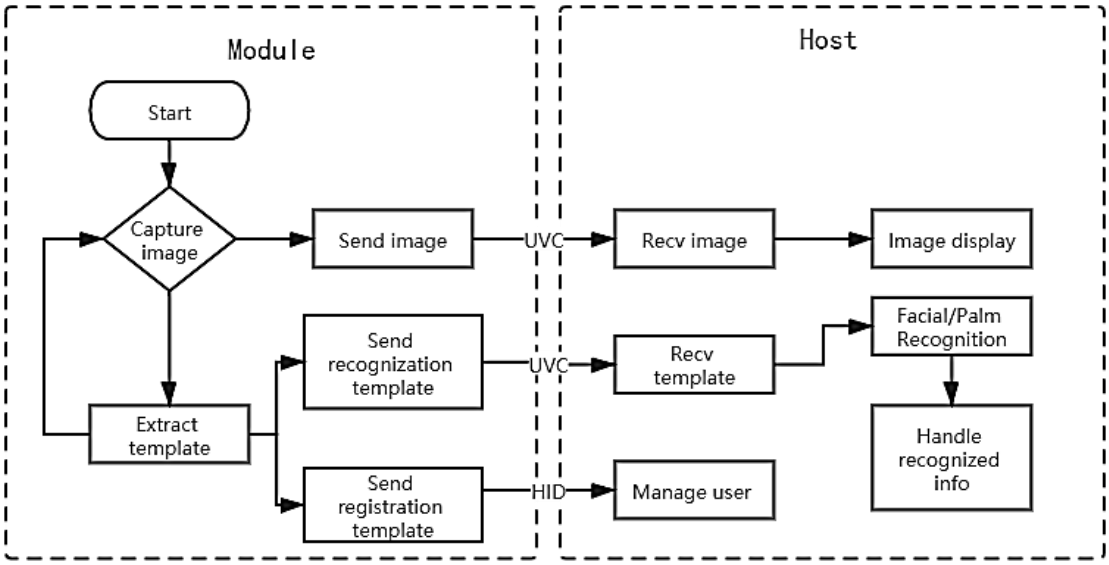
the development cycle, facilitates the interpretation of various application scenarios, and enhances productivity.

2.2.1 UVC Image Acquisition and Transmission

- The module supports two-channel image transmission of visible light and near-infrared, and both the collection and transmission process are completed on module.
- The user collects images through AMTCameraManager or UsbCameraManager.
- These captured images could be used for display, face recognition, palm recognition and other purposes.
- The module now supports two resolutions of 480 x 640 and 720 x 1280 and an output configuration of up to 30FPS.
- The image transmission adopts the standard UVC protocol and transmits the visible light image and the near-infrared image through the corresponding UVC ports, and the developers can select the required port settings according to the image requirements.

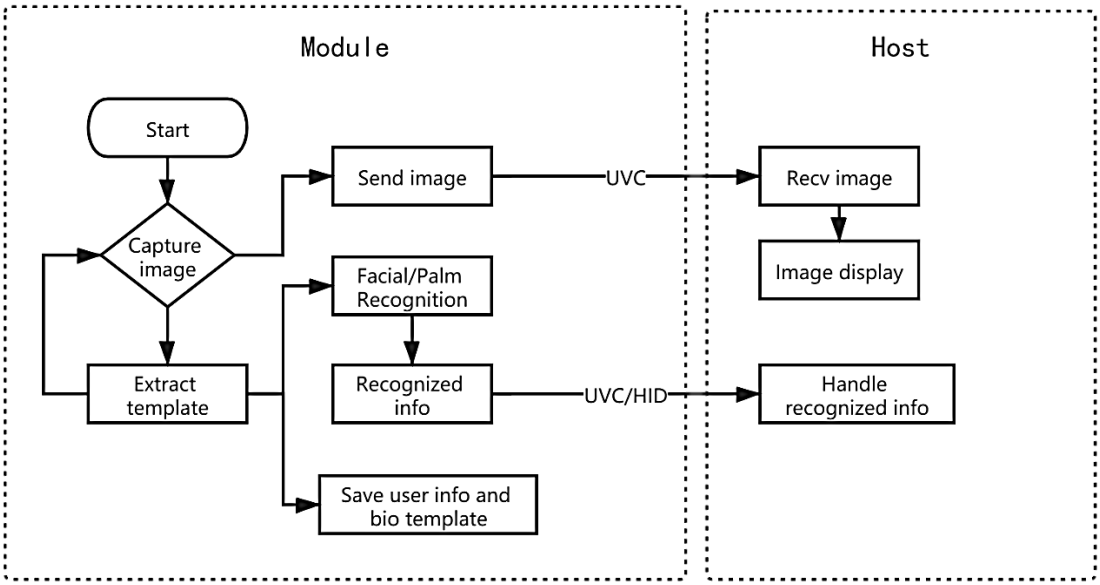
2.2.2 Match-on-Host and User Management

- If template matching and user management are performed on the host side, the module can push the biometric template to the host side via UVC data once the feature template has been extracted on the module side.
- Host side can complete the subsequent template verification through [AMTFaceMatch] and [AMTPalmMatch].
- This method is available in high-performance devices and application scenarios that have special needs for template storage.
- The following images describe the Match-on-Host and user management design.



2.2.3 Match-on-Module and User Management

- If the developer prefers to perform verification and user management within the module, then the module will generate the output of the verification result through the UVC protocol.
- And after the verification process completed for the client application to call, it produces the recognition result through the Polling Recognition Result.
- This method can minimize the computing resource consumption of the client application platform processor and is particularly suitable for integrating face or palm recognition functions on low-performance embedded platforms.
- The design of Match-on-Module and user management is as follows.



2.3 Fast Integration

Users can refer to this section to quickly integrate SDK into applications to realize development functions.

2.3.1 SDK File

- Copy the amtmultibio.aar to the app libs directory.
- Add the jniLib dependent library to the android node in the gradle file.

```
sourceSets.main
    {
            jniLibs.srcDir 'libs'
            jni.srcDirs = []
    }
```

- Add the dependencies node.

```
dependencies
    {
            implementation(name: 'amtmultibio', ext: 'aar')
    }
```

File Name	Description
amtmultibio.aar	AMTMultiBio SDK

2.3.2 Application Permissions

The SDK needs to meet the following conditions to run normally. Please ensure that the following requirements comply.

- Add the following code to the node in AndroidManifest.xml

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature
        android:name="android.hardware.usb.host"
        android:required="true" />
```

- In the android node using build.gradle, add the following configuration to disable the

compression of the so library in the sdk, otherwise there will be abnormalies.

```
packagingOptions {
        doNotStrip "*/armeabi-v7a/*.so"
        doNotStrip "*/arm64-v8a/*.so"
}
```

Note:

- Make sure that the device/dev/video* node has both read and write permissions if AMTCameraManager is used.
- Do not need to implement another USB device plugging and listening If USBCameraManager is used, because USBCameraManager already implements its own plugging and listening.

2.3.3 USB Information

Device Name	Vendor ID	Product ID
AMT-FAPVS-21	0x34C9	0x3141
AMT-FAPVR-21	0x34C9	0x3181
AMT-FAPVS-21 (Upgrading image mode)	0x34C9	0x3000

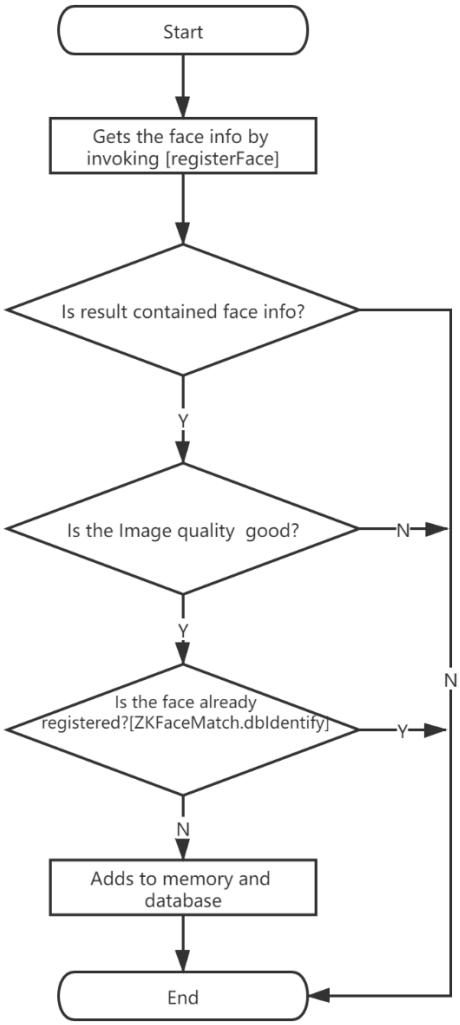
2.4 Programming Guide

Users can quickly understand the registration and verification process of the user's face and palm on the Host side through **Match-on-Host** and **User Management**.

Users can quickly understand the process of internal staff management and internal registration verification of the module by referring to **Internal Registration Verification Process**.

2.4.1 Match-on-Host and User Management

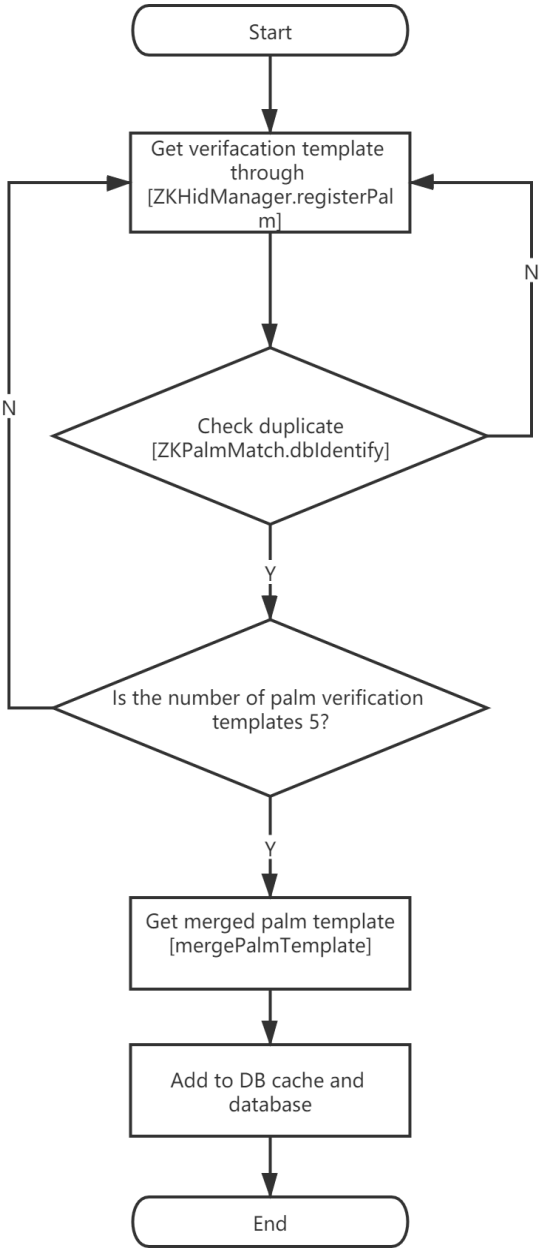
Face Registration Process Flow



Process Description

- The **registerFace** function passes an image containing a visible face or captures a face image directly through the module.
- The returned face information contains the following attributes:
  1. Angle of the face (recommended yaw<=25, pitch<=25, roll<=25)
  2. Size (recommended faceWidth>=100, faceHeight>=100)
  3. Quality (recommended >=0.8)
- This information determines the quality of the captured face.
- The AMTFaceMatch.dbIdentify function checks for the duplicate template.
- And if there is no duplicate, then the newly captured template is added to the algorithm cache by AMTFaceMatch.dbAdd function.

Palm Registration Process Flow

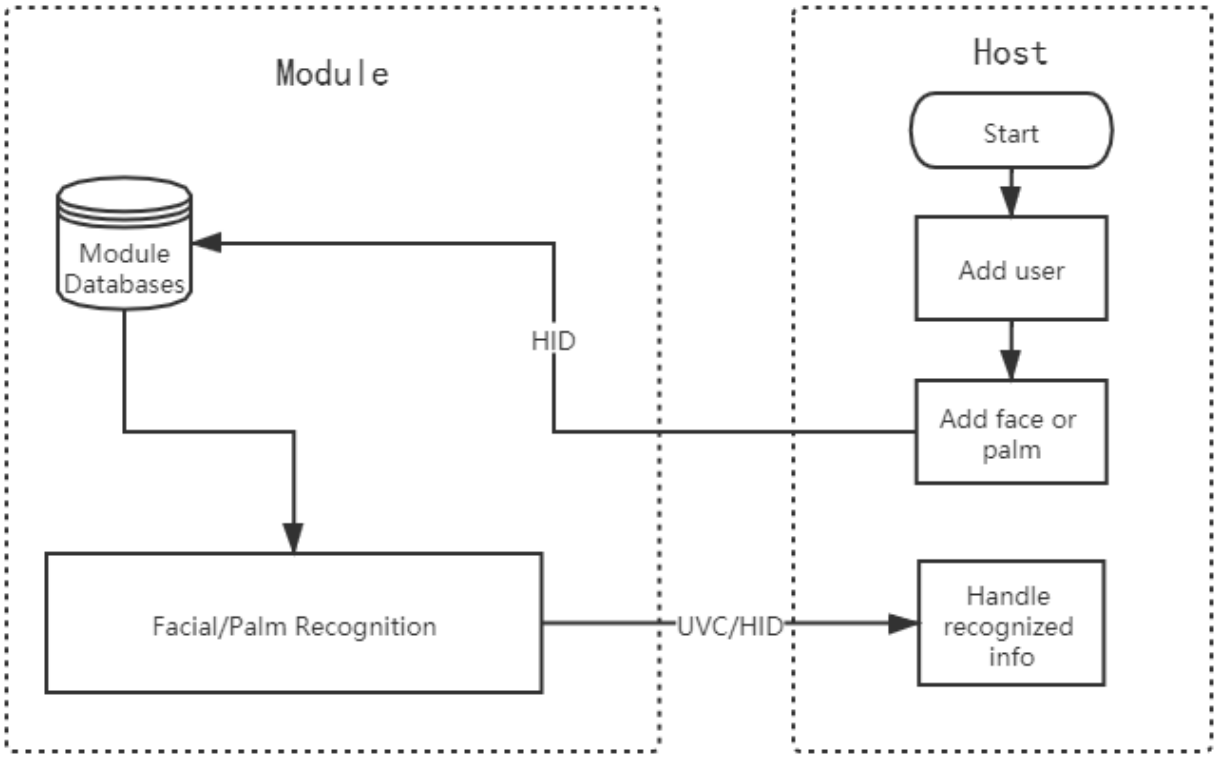


Process Description

- The **RegisterPalm** function can import 8-bit grayscale images or directly collect palm grayscale images through the module.
- The first five collected templates contain the comparison templates that are filtered via AMTPalmMatch.dbIdentify function.
- After collecting the five templates, call the mergePalmTemplate API to get the merged registration template.
- This merged registration template is then added to the algorithm cache via AMTPalmMatch.dbAdd function.

2.4.2 Match-on-Module Process

- Here the user can directly register the face/palm inside the module, where the personnel information and the biometric template will immediately get stored inside the module.
- And the identified personnel information is obtained through **UVC** or **Polling Recognition Result**.
- The flowchart of the internal registration of the module is as follows:



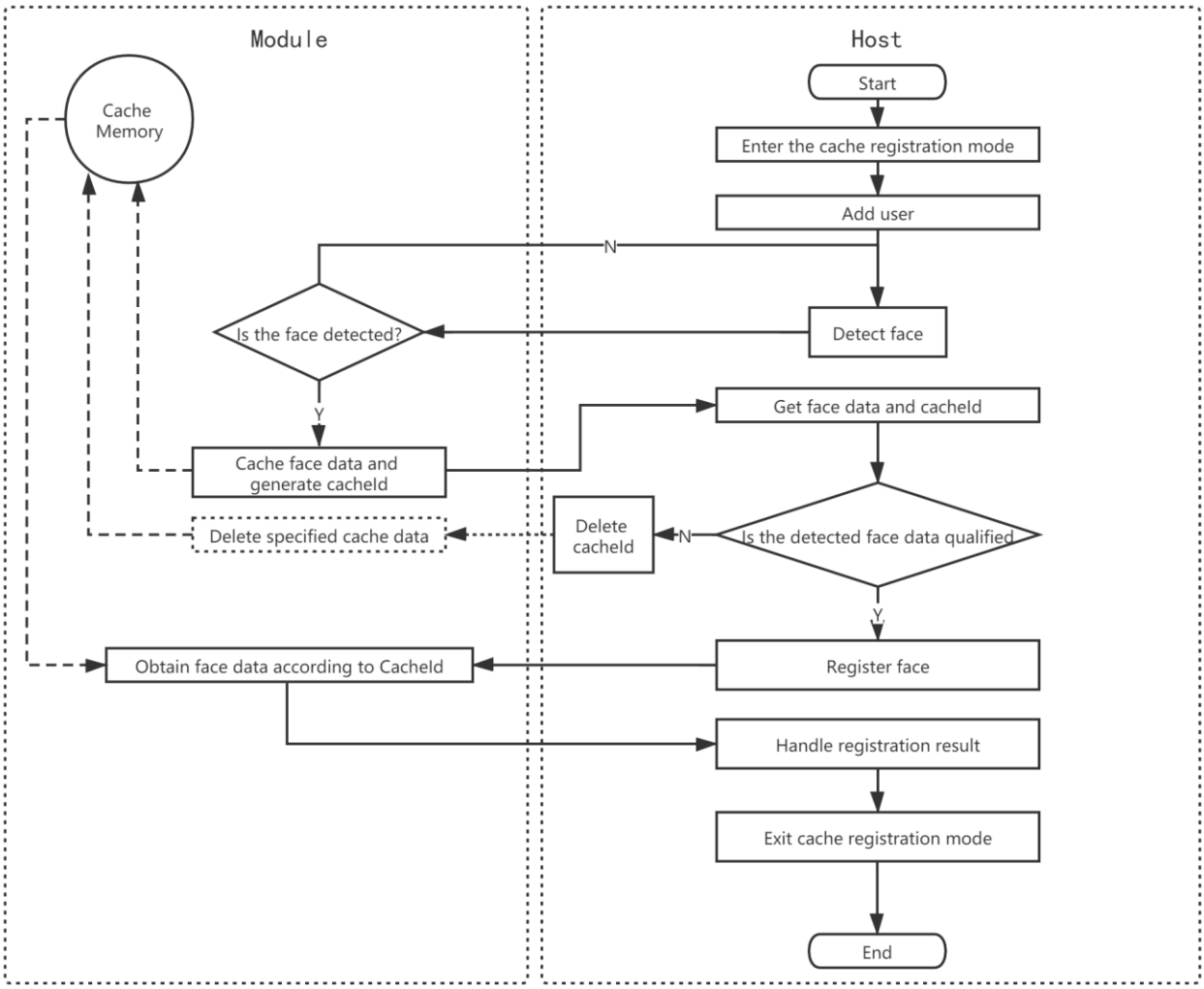
Note: If using polling recognition results, it is required to suspend HID polling when using the HID command to obtain other data or else it will affect the acquisition of the other data.

Process Description

- To use the Match-on-Module mode, you need to add personnel information on the Host side first. Face and palm registration can upload photos through the host side or register directly through the **Cache Registration** mode.

Face Cache Registration

When using the Match-on-Module mode, the face can be registered through the internal CacheId in cache registration mode. Refer to the following process:

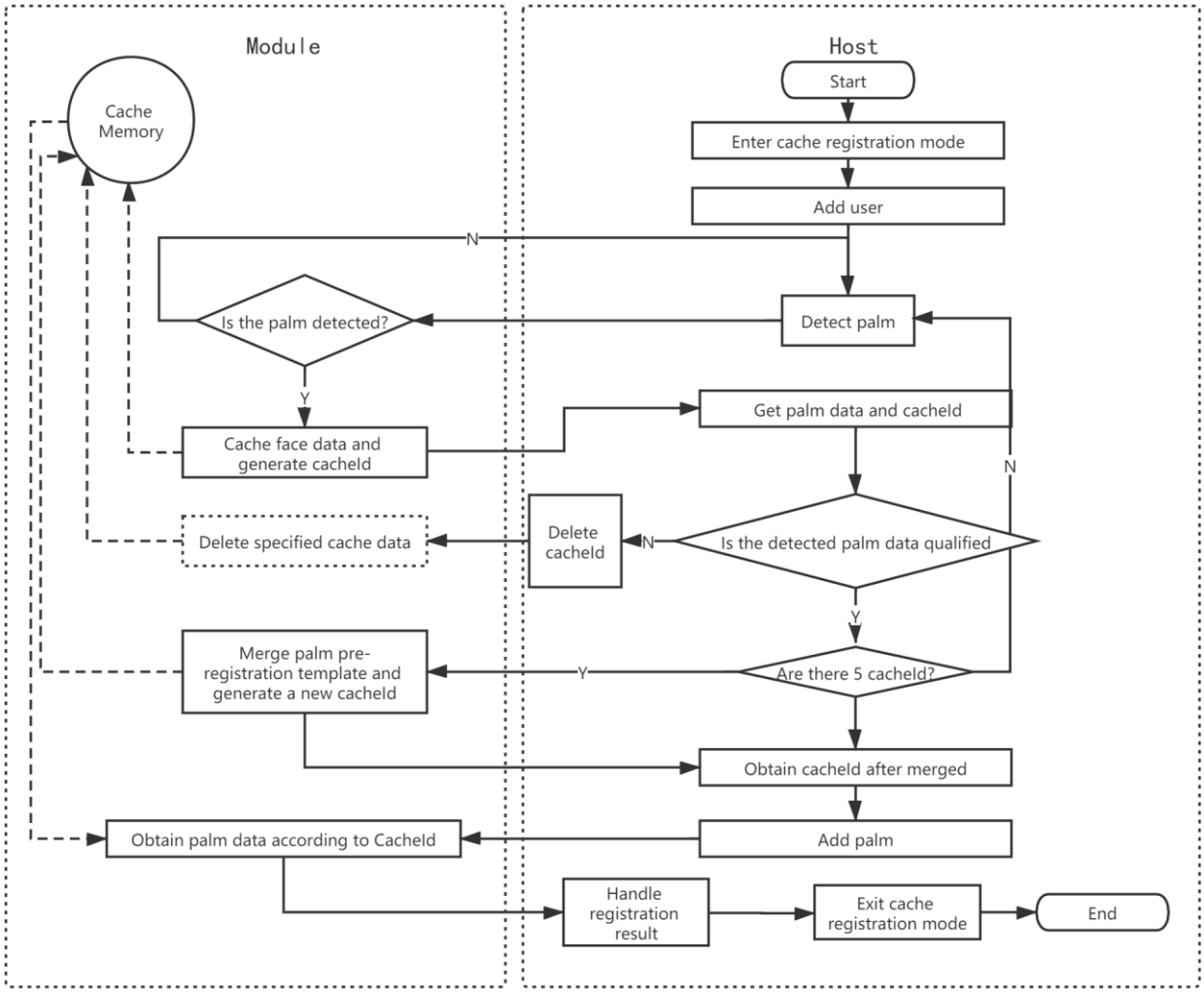


Process Description

- When registering a face using the cache registration mode, first enter the [Cache Registration] mode, and then get the CacheId.
- After the face registration is complete, it is necessary to exit the [Cache Registration] mode or else the face and palm cannot be compared.

Palm Cache Registration

When using the Match-on-Module mode, the palm can be registered through the internal CacheId in cache registration mode. Refer to the following process:



Process Description

- When registering a palm using the cache registration mode, first enter the [Cache Registration] mode, and then get the CacheId.
- After the palm registration is complete, it is necessary to exit the [Cache Registration] mode or else the face and palm cannot be compared.

3 SDK API Description

3.1 UVC Image Capture API with Root Privileges

AMTCameraManager is a V4L2-based video stream capture API for use on devices with root privileges. If the device does not have root privileges, please refer to the section **3.2 UVC Image Capture Interface 2** to acquire UVC video streams and custom data.

Function List

Interface	Description
VideoData	Video data structure, including image data, etc.
CustomData	Custom data structure, including face, palm, and other information
CameraDataCallback	UVC camera data callback interface.
open	Turns on the UVC device with specified parameters
close	Turns off the UVC device
init	Turns on the UVC device with the specified parameters and sets the UVC data Callback API
register	Registers the USB device, attach and detach monitor, and requests access permission
unregister	Unregisters the USB device monitor
release	Recycles the available resources
getCapWidth	Gets the UVC image width
getCapHeight	Gets the UVC image height
startCapture	Initiates to capture images from UVC device
stopCapture	Stops capturing images from UVC device
setCallback	Sets the UVC data callback API
removeCallback	Removes the UVC data callback API
getDevices	Gets the UVC device address index
getDevType	Gets the UVC device type according to the UVC address index.

Note: The init, register, unregister and release interfaces are for automatic access under USB connection. If the device is through V4L2, you don't need to pay attention to these four interfaces.

VideoData

Function Syntax

```
public class VideoData
        {
                public byte[] buff;
                public int size;
                public int frame_index;
                public int ori_size;
        }
```

Parameter Description

Parameter	Description
buff	Original Image data
size	Original Image data size
frame_index	UVC Frame Index
ori_size	Original JPEG data length

Remarks

- The videoData object is used to customize the UVC frame image data including JPEG image and other information.
- Click here to view the function list.

CustomData

Function Syntax

```
public class CustomData
{
        public int frame_index;
        public int width;
        public int height;
        public byte[] data;
}
```

Parameter Description

Parameter	Description
frame_index	Refers the detected image frame number
width	Image detection width
height	Image detection height
data	Original custom JSON data

Remarks

- The Custom UVC frame JSON format data includes Face Tracking, Face Extraction Template, Palm Detection, Palm Extraction, and other information.
- For JSON data definition, please refer Face tracking and recognition and Palm recognition.
- Click here to view the function list.

CameraDataCallBack

This function is used for UVC camera data callback.

Function	Description
onFrameDataRecv()	UVC image data callback
onCustomDataRecv()	Custom UVC data callback

onFrameDataRecv

Function Syntax

void onFrameDataRecv(CustomFrameData videoData)

Description

UVC image frame data callback

Parameter Description

Parameter	Description
videoData	**In/Out**: CustomFrameData object

onCustomDataRecv

Function Syntax

void onCustomDataRecv(CustomData customData);

Description

UVC image frame data callback

Parameter Description

Parameter	Description
customData	**In/Out:** CustomData object

Remarks

- Click here to view the function list.

open

Function Syntax

public boolean open
(
String devAddress,
int iw,
int ih,
int fps
)

Description

Turns on the UVC device with the specified parameters.

Parameter Description

Parameter	Description
devAddress	**In:** UVC Device Address
iw	**In:** Needs to set the UVC image width
ih	**In:** Needs to set the UVC image height
fps	**In:** This indicates the number of acquisition frames to be set. Currently, the supported numbers are 25 and 30.

Returns

True	Successfully opened
False	Failed to open

Remarks

- Click here to view the function list.

close

Function Syntax

public void close()

Description

Turns off the opened UVC device

Remarks

- Click here to view the function list.

init

Function Syntax

```
public void init
    (
        Context context,
        final int pid,
        final int vid,
        final int width,
        final int height,
        CameraDataCallback cameraDataCallback
    )
```

Description

Turns on the UVC device with the specified parameters and sets the UVC data Callback API.

Parameter Description

Parameter	Description
context	**In:** Android context
pid	**In:** Device product ID
vid	**In:** Device vendor ID
width	**In:** Needs to set the UVC image width
height	**In:** Needs to set the UVC image height
cameraDataCallback	**In/out:** CameraDataCallback callback implementation object

Returns

void

Remarks

- Click here to view the function list.

register

Function Syntax

public void register()

Description

Registers the USB device, attach and detach monitor, and requests access permission.

Returns

void

Remarks

- Click here to view the function list.

unregister

Function Syntax

public void unregister()

Description

Unregisters the USB device monitor.

Returns

void

Remarks

- Click here to view the function list.

release

Function Syntax

public void release()

Description

Recycles the available resources.

Returns

void

Remarks

- Click here to view the function list.

GetCapWidth

Function Syntax

public int GetCapWidth()

Description

Gets the image width of the current UVC device.

Returns

Image width

Remarks

- Click here to view the function list.

GetCapHeight

Function Syntax

public int GetCapHeight()

Description

Gets the image height of the current UVC device.

Returns

Image height

Remarks

- Click here to view the function list.

StartCapture

Function Syntax

public boolean StartCapture()

Description

Initiates to capture images from the current UVC device.

Returns

True	Image successfully taken
False	Capture failed

Remarks

- Click here to view the function list.

StopCapture

Function Syntax

public void StopCapture()

Description

Stops capturing images from the current UVC device.

Remarks

- Click here to view the function list.

setCallback

Function Syntax

public void setCallback(CameraDataCallback callback)

Description

Sets the UVC data callback API.

Parameter Description

Parameter	Description
callback	**In/Out**: CameraDataCallback callback implementation object

Remarks

- Click here to view the function list.

removeCallback

Function Syntax

public void removeCallback()

Description

Removes the UVC data callback API.

Remarks

- Click here to view the function list.

getDevices

Function Syntax

public int[] getDevices()

Description

Gets all the UVC device address indexes.

Returns

Returns all the UVC device address indexes.

Remarks

- Click here to view the function list.

getDevType

Function Syntax

public int getDevType(int index)

Description

Gets the UVC device type according to the UVC device address index.

Returns

-1	Error
0	Unknown Device
1	RGB UVC Camera
2	NIR UVC Camera

Remarks

- Click here to view the function list.

3.2 UVC Image Capture API without Root Privileges

UsbCameraManager is a UVC streaming mode-based video streaming capture API that can be used on any device that supports USB HOST mode without root access.

Note: It is not necessary to implement USB device plugging and listening when using UsbCameraManager, which is already implemented inside USBCameraManager.

Function List

Interface	Description
init	Turns on the UVC device with the specified parameters and sets the UVC data Callback API

register	Registers the USB device, attach and detach monitor, and requests access permission
unregister	Unregisters the USB device monitor
switchCamera	Switch camera image streams
release	Recycles the available resources
closeAllCamera	Turn off all open cameras

Init

Function Syntax

```
public void init
(
        Application application,
        final int vid,
        final int pid,
        final int width,
        final int height,
         CameraDataCallback cameraDataCallback
)
```

Parameter Description

Parameter	Description
application	**In:** Android context
vid	**In:** Device product ID
pid	**In:** Device vendor ID
`width`	**In:** Needs to set the UVC image width
`height`	**In:** Needs to set the UVC image height
`cameraDataCallback`	**In/out:** CameraDataCallback callback implementation object

Remarks

- Click here to view the function list.

register

Function Syntax

```
public void register()
```

Description

Registers the USB device, attach and detach monitor, and requests access permission.

Returns

void

Remarks

- Click here to view the function list.

unregister

Function Syntax

```
public void unregister()
```

Description

Unregisters the USB device monitor.

Returns

void

Remarks

- Click here to view the function list.

switchCamera

Function Syntax

```
public void switchCamera()
```

Description

Switch the current camera image stream. The module has 2 video streams by default.

Returns

void

Remarks

- Click here to view the function list.

release

Function Syntax

```
public void release()
```

Description

Recycles the available resources.

Returns

void

Remarks

- Click here to view the function list.

closeAllCamera

Function Syntax

```
public void closeAllCamera ()
```

Description

Turn off all open cameras.

Returns

void

Remarks

- Click here to view the function list.

3.3 HID Communication Interface

AMTHidManager is a HID communication interface, mainly used for reading, setting parameters, user management, firmware upgrades, module image upgrades, etc.

Note: The usage of HIDManager requires the implementation of USB device broadcast plugging and listening on the application side.

Function List

API Feature	Description

open	Turns on the HID device
close	Turns off the HID device
getConfig	Gets the specified configuration information
setConfig	Sets the specified configuration
registerFace	Face Registration
registerPalm	Palm Registration
mergePalmTemplate	Gets the merged palm registration template
snapShot	Snapshot
sendFile	Sends the file
reboot	Reboots the device
registerUsbReceiver	Registers the USB permission broadcast
unregisterUsbReceiver	Unregisters the USB permission broadcast
manageModuledata	Manages the module internal data, including user management, biological templates, matching records, etc.
pollMatchResult	Polls the recognition result

open

Function Syntax

public int open(Context context)

Description

Establishes the connection with the HID device.

Parameter Description

Parameter	Description
context	**In:** Android context

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- Click here to view the function list.

close

Function Syntax

public int close()

Description

Disconnects from HID device

Parameter

None

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- Click here to view the function list.

getConfig

Function Syntax

```
public int getConfig
        (
                int type,
                byte[] result,
                int[] size
        )
```

Description

Gets the configuration information of the specified type

Parameter Description

Parameter	Description
type	**In** 1 COMMON_CONFIG 2 CAPTURE_FILTER_ CONFIG 3 MOTION_DETECT_CONFIG

	4 PALM_CONFIG 5 DEVICE_INFORMATION 6 DEVICE_TIME
result	**Out:** Returns the JSON data. It is recommended to allocate 20*1024* bytes
size	**In:** Allocated result size. If it is **0**, the default size is used. It is recommended to allocate according to the actual situation.
	Out: Returns the actual result size

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For returned JSON data, please refer Get and set configuration parameters.
- Click here to view the function list.

setConfig

Function Syntax

```
public int setConfig
        (
                int type,
                byte[] json,
                int size
        )
```

Description

Sets the configuration parameters of the specified type of the face module

Parameter Description

Parameter	Description
type	**In:** 1 COMMON_CONFIG 2 CAPTURE_FILTER_ CONFIG 3 MOTION_DETECT_ CONFIG 4 PALM_ CONFIG

	5 RESTORE_FACTORY 6 DEVICE_TIME
json	**In:** Corresponding configured JSON data
size	**In:** JSON data size

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer Get and set configuration parameters.
- Click here to view the function list.

registerFace

Function Syntax

public int registerFace

(

byte[] param,

byte[] result,

int[] size

)

Description

Uploads the photo for face registration.

Parameter Description

Parameter	Description
param	**In:** Face registration data, please refer to the data format
result	**Out:** It returns the face registration data. It is recommended to allocate at least 20*1024* bytes
size	**In:** Allocated result size. If it is **0**, the default size is used It is recommended to allocate according to the actual situation.
	Out: Size of the returned face registration data

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer to Host side face registration process and [Face service-related functions].
- Click here to view the function list.

registerPalm

Function Syntax

public int registerPalm

(

byte[] param,

byte[] result,

int[] size

)

Description

Gets the palm pre-registration template

Parameter Description

Parameter	Description
param	**In:** For palm registration data, please refer to the data definition
result	**Out:** Allocate at least 200*1024*bytes for the returned JSON data of preregistered template and verification/identification template
size	**In:** The size of passed result. If it is **0**, the default size is used. It is recommended to allocate according to the actual situation.
	Out: The actual size of the returned result

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer to Host side palm registration process and Palm service-related functions.
- Click here to view the function list.

mergePalmTemplate

Function Syntax

public int mergePalmTemplate

(

byte[] mergeTemplate,

int[] size

)

Description

Gets the merged palm template.

Parameter Description

Parameter	Description
mergeTemplate	**Out:** Returns the merged palm registration template
size	**In**: Size of the allocated mergeTemplate. If it is **0**, the default size is used. It is recommended to allocate according to the actual situation.
	Out: Returns the actual mergeTemplate size

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer Merge palm pre-registration template.
- Click here to view the function list.

snapShot

Function Syntax

public int snapShot

(

int snapType,

byte[] result,

int[] size

)

Description

Captures a picture

Parameter Description

Parameter	Description
snapType	**In:** SnapType.SNAP_GRAY 8-bit grayscale image SnapType.SNAP_RGB Color image
result	**Out:** It returns the captured image JSON data. It is recommended to allocate at least 2*1024*1024 bytes
size	**In:** Size of the allocated result. If it is 0, the default size is used, please allocate according to the actual situation.
	Out: It returns the result data length

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- Click here to view the function list.

sendFile

Function Syntax

```
public int sendFile
(
    string filePath,
    SendFileProgressListener sendFileProgressListener
)
```

Description

Sends files to the module

Parameter Description

Parameter	Description
filePath	**In:** File Path
sendFileProgressListener	**In:** File sending status callback

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- Click here to view the function list.

reboot

Function Syntax

public int reboot(int mode)

Description

Reboots the device

Parameter Description

Parameter	Description
mode	**In:** 3 Normal reboot 4 Reboot to upgrade mode

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- Click here to view the function list.

registerUsbReceiver

Function Syntax

public void registerUsbReceiver()

Description

Registers the HID device broadcast to obtain USB permissions

Remarks

- Click here to view the function list.

unregisterUsbReceiver

Function Syntax

public void unregisterUsbReceiver()

Description

Unregisters the HID device broadcast

Remarks

- Click here to view the function list.

manageModuleData

Function Syntax

public int manageModuleData

(

int manageType,

byte[] param,

byte[] result,

int[] resultSize

)

Description

Module internal data management, data user data, matching records, biological templates, etc.

Parameter Description

Parameter	Description
manageType	**In:** 1 ADD_PERSON 2 DEL_PERSON 3 CLEAR 4 GET_PERSON 5 QUERY_ALL_PERSON 6 QUERY_STATISTICS 7 ADD_FACE 8 ADD_FACE_REG 9 DETECT_FACE_REG 10 REG_START 11 REG_END 12 DETECT_PALM_REG 13 ADD_PALM

	14 ADD_PALM_REG 15 MERGE_PALM_REG 16 DEL_FACE_CACHE_ID 17 DEL_PALM_CACHE_ID 18 ATT_RECORD_COUNT 19 EXPORT_ATT_RECORD 20 CLEAR_ATT_RECORD
param	**In:** Data passed in when managing module data
result	**Out:** Results returned when managing module data
resultSize	**In:** Size of the allocated result. If it is 0, the default size is used. It is recommended to allocate according to the actual situation.
	Out: The actual result size of the management module data.

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer to [Module data management_snapShot].
- Click here to view the function list.

pollMatchResult

Function Syntax

```
public int pollMatchResult
(
        byte[] result,
        int[] size
)
```

Description

Polls recognition result

Parameter Description

Parameter	Description
result	**Out:** Face/palm recognition result, please allocate at least 40*1024 byte

size	**In:** Size of the allocated result. If it is 0, the default size is used. It is recommended to allocate according to the actual situation. **Out:** Length of result data actually returned

Returns

Returns 0 for success

Remarks

- Please refer Appendix 1 for further details.
- For JSON data, please refer to [Polling recognition result_snapShot].
- Click here to view the function list.

3.4 AMTFaceMatch

AMTFaceMatch is a face-matching algorithm interface used to achieve 1:1 and 1:N face verification/identification functions.

Function List

API Feature	Description
init	Initializes the face library verification/identification process
terminate	Ends the face library verification/identification process
verify	1:1 Face Verification
dbAdd	Adds the face template to the Database
dbDel	Deletes the specified face template from the Database
dbClear	Clears the Database
dbCount	Gets the total count of the face template from the Database
dbVerify	Verifies the captured template with the template of that specified ID stored in the database and returns the verification value.
dbIdentify	Identifies the captured template against all the stored templates in the database and returns the identification value.

init

Function Syntax

public static int init(long[] context)

Description

Initializes the face library verification/identification process.

Parameter Description

Parameter	Description
context	**In/Out:** Face library pointer

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

terminate

Function Syntax

public static int terminate(long context)

Description

Ends the face library verification/identification process.

Parameter Description

Parameter	Description
context	**In:** Face library pointer

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

verify

Function Syntax

```
public static int verify
        (
                long context,
                byte[] template1,
                byte[] template2,
                float[] score
        )
```

Description

1:1 Face Verification

Parameter Description

Parameter	Description
context	**In:** Face library pointer
template1	**In:** Registration template
template2	**In:** Verification template
score	**Out:** Returns the verification score

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

dbAdd

Function Syntax

```
public static int dbAdd
        (
                long context,
                byte[] id,
                byte[] template
        )
```

Description

Adds the face template to the Database

Parameter Description

Parameter	Description
context	**In:** Face library pointer
id	**In:** Face ID
template	**In:** Registration template

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

dbDel

Function Syntax

```
public static int dbDel
        (
                long context,
                byte[] id
        )
```

Description

Deletes the specified face template from the Database

Parameter Description

Parameter	Description
context	**In:** Face library pointer
id	**In:** Face ID

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
-
- Click here to view the function list.

dbClear

Function Syntax

public static int dbClear(long context)

Description

Clears the Database

Parameter Description

Parameter	Description
context	**In:** Face library pointer

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.

- Click here to view the function list.

dbCount

Function Syntax

public static int dbCount

(

long context,

int[] size

)

Description

Gets the total count of the face template from the Database

Parameter Description

Parameter	Description
context	**In:** Face library pointer
size	**Out:** It returns the actual size of the face library

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

dbVerify

Function Syntax

public static int dbVerify

(

long context,

byte[] template,

byte[] id,

float[] score

)

Description

Verifies the captured template with the template of that specified ID stored in the database and returns the verification value.

Parameter Description

Parameter	Description
context	**In:** Face library pointer
template	**In:** Registration template
id	**In:** Face ID
score	**Out:** Returns the verification score

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

dbIdentify

Function Syntax

public static int dbIdentify

(

long context,

byte[] template,

byte[] id,

float[] score

)

Description

Identifies the captured template against all the stored templates in the database and returns the identification value.

Parameter Description

Parameter	Description
context	**In:** Face library pointer

template	**In:** Identification template
id	**Out:** Face ID
score	**Out:** Returns the identification score

Returns

Error Code

Remarks

- Please refer Appendix 2 for further details.
- Click here to view the function list.

3.5 AMTPalmMatch

AMTPalmMatch is a palm comparison algorithm interface used to implement 1:1 and 1:N verification/identification functions on the Host side.

Function List

API Feature	Description
init	Initializes the palm verification/identification process
terminate	Stops the palm verification/identification process
mergeTemplates	Merges the pre-registered templates and returns a registered template
verify	1:1 Palm Verification
dbAdd	Adds the palm template to the Database
dbDel	Deletes the specified palm template from the Database
dbClear	Clears the Database
dbCount	Gets the total count of the palm templates from the Database
dbVerify	Verifies the captured template with the template of that specified ID stored in the database and returns the verification value
dbIdentify	Identifies the captured template against all the stored templates in the database and returns the identification value

init

Function Syntax

public static int init(long[] context)

Description

Initializes the palm verification/identification process.

Parameter Description

Parameter	Description
context	**In/Out:** Palm library pointer

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

terminate

Function Syntax

public static int terminate(long context)

Description

Stops the palm verification/identification process

Parameter Description

Parameter	Description
context	**In:** Palm library pointer

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

mergeTemplates

Function Syntax

```
public static int mergeTemplates
        (
                long context,
                byte[][] preRegTemplates,
                int mergedCount,
                byte[] regTemplate,
                int[] regTemplateLength
        )
```

Description

Combines the pre-registered palm templates into one registration template

Parameter Description

Parameter	Description
context	**In:** Palm library pointer

preRegTemplates	**In:** Pre-registration template
mergedCount	**In:** Number of pre-registered templates
regTemplate	**Out:** Returns one registration template
regTemplateLength	**In:** The size of the allocated registration template space
	Out: Returns the actual registration template size

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

verify

Function Syntax

public static int verify

(

long context,

byte[] template1,

byte[] template2,

int[] score

)

Description

1:1 Palm Verification

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
template1	**In:** Registration palm template
template2	**In:** Verification palm template
score	**Out:** Returns the verification score

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbAdd

Function Syntax

public static int dbAdd

(

long context,

byte[] id,

byte[] template

)

Description

Adds the palm template to the Database

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
id	**In:** Palm ID
template	**In:** Registration template

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbDel

Function Syntax

```
public static int dbDel
        (
                long context,
                byte[] id
        )
```

Description

Deletes the palm template from the Database

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
id	**In:** Palm ID

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbClear

Function Syntax

```
public static int dbClear(long context)
```

Description

Clears the Database

Parameter Description

Parameter	Description
context	**In:** Palm library pointer

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbCount

Function Syntax

public static int dbCount

(

long context,

int[] size

)

Description

Gets the total count of the palm templates from the Database

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
size	**Out:** Returns the actual size of the palm library

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbVerify

Function Syntax

public static int dbVerify

(

long context,

byte[] template,

byte[] id,

int[] score

)

Description

Verifies the captured template with the template of that specified ID stored in the database and returns

the verification value

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
template	**In:** Registration palm template
id	**In:** Palm ID
score	**Out:** Returns the verification score

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

dbIdentify

Function Syntax

public static int dbIdentify

(

long context,

byte[] template,

byte[] id,

int[] score

)

Description

Identifies the captured template against all the stored templates in the database and returns the identification value

Parameter Description

Parameter	Description
context	**In:** Palm library pointer
template	**In:** Identification template
id	**In:** Palm ID
score	**Out:** Returns the identification score

Returns

Error Code

Remarks

- Please refer Appendix 3 for further details.
- Click here to view the function list.

4 Data Communication

The data exchange between the module and the host side supports both UVC and HID methods. UVC is mainly responsible for Big data transmission such as image transmission, and HID is responsible for sending commands and receiving small data such as configuration.

4.1 General Json Data

General JSON data is JSON data that has the same data structure and frequently used during data interaction.

4.1.1 Image

Image data is mainly used for registration and returns the registered photos when using the images of faces and palms.

The JSON data for image is as follows:

Function Syntax

```
{
        "image": {
                "bioType": "face",
                "data": "/9j/4AAQSkZJRgABA",
                "format": "jpeg",
                "width": 720,
                "height": 1280
        }
}
```

Parameter Description

Parameter	Is it Required	Description
bioType	Yes	The type of image information returned. • Face: "face" • Palm: "palm"
data	Yes	The returned image base64 data.
format	Yes	The returned image format. jpeg, gray, etc.
width	Yes	The returned image width.
height	Yes	The returned image height.

4.1.2 CacheId

The CacheId is primarily for [Cache Registration], and the registration method can refer to [Face Cache Registration] and [Palm Cache Registration].

The JSON data for CacheId is as follows:

Function Syntax

```
{
        "cacheId": {
                "bioType": "face",
                "data": 1
        }
}
```

Parameter Description

Parameter	Is it Required	Description
bioType	Yes	The type of image information returned. • Face: "face" • Palm: "palm"
data	Yes	The data Id stored inside the module, can be used to add the user's face, palm, etc.

4.1.3 Feature

Biological template data can be used for face and palm recognition and registration. The biological template json data is as follows:

Function Syntax

```
{
        "feature": {
                "bioType": "face",
                "data": "xxxx",
                "size": 1024
        }
}
```

Parameter Description

Parameter	Is it Required	Description
bioType	Yes	The type of image information returned. • Face: "face" • Palm: "palm"

data	Yes	The returned template base64 data
size	Yes	The returned template size

4.1.4 Attribute

The face attribute data used to provide custom functions, such as prompting different voices according to gender and age.

The JASON data for face attribute is as follows.

Function Syntax

```
{
        "attribute": {
                "age": 29,
                "beauty": -1,
                "cap": 0,
                "expression": 0,
                "eye": -1,
                "gender": 1,
                "glasses": -1,
                "mouth": -1,
                "mustache": 1,
                "respirator": 0,
                "skinColor": 0,
                "smile": -1
        }
}
```

Parameter Description

Parameter	Is it Required	Description
age	Yes	Age [0-100]
beauty	Yes	Beauty 0: no,1: yes
cap	Yes	Hat 0 No hat 1 Safety helmet 2 Chef hat 3 Student hat 4 Helmet 5 Taoism hat 6 Kerchief 7 Others 8 Unknown
expression	Yes	Expression:

		0 calm 1 happy 2 angry 3 sorrow 4 surprise 5 scared 6 disgust 7 yawn
eye	Yes	Whether to close eyes 0: no,1: yes
gender	Yes	Gender 0: male,1: female
glasses	Yes	Whether to wear glasses 0: no, 1: yes
mouth	Yes	Whether to open mouth 0: no,1: yes
mustache	Yes	Whether to have a mustache 0: no, 1: yes
respirator	Yes	Whether to have a mask 1: yes, other: no
respiratorLevel	Yes	Mask Covering Level 0 None 1 Full cover 2 Mouth not covered 3 Nose not covered
skinColor	Yes	Skin color 0 Yellow 1 White 2 Black 3 Brown 4 Unknown
smile	Yes	Smile 0 Calm 1 Happy 2 Angry 3 Sorrow 4 Surprise 5 Scared 6 Disgust, yawn

4.1.5 Identify

Identify is the recognition result returned by UVC or [Polling Recognition Result] during internal comparison.

The JSON data is as follows:

Function Syntax

```
{
        "identify": [{
```

```
                "groupId": "",
                "name": "1180665",
                "personId": "1180665",
                "similarity": 0.9328765869140625,
                "userId": "1180665"
        }]
}
```

Parameter Description

Parameter	Is it Required	Description
groupId	Yes	The Id of the group the module user belongs to
name	Yes	The recognized user name
personId	Yes	The identified user personId, generally the same as userId
similarity	Yes	The similarity of face/palm
userId	Yes	The identified user userId, generally the same as personId

4.1.6 Liveness

The face liveness data determines whether it is a photo or a video incident.

The face liveness JSON data is as follows:

Function Syntax

```
        {
        "liveness": {
                "irFrameId": 10339,
                "liveness": 2,
                "livenessMode": 12,
                "livenessScore": 0.91753107309341431,
                "quality": 0.90849864482879639
        }
}
```

Parameter Description

Parameter	Is it Required	Description
irFrameId	Yes	Corresponding NIR camera frame index
liveness	Yes	0 Liveness detection is not turned on 1 Dummy 2 Real people 11 No image data 30 Face detection failed

		31 IoU exception
livenessMode	Yes	Liveness mode, 11 Dual Cameras 12 NIR Camera 13 RGB Camera
livenessScore	Yes	Liveness score
quality	Yes	The quality of face tracking

4.1.7 Landmark

The key point coordinates of the face in Landmark, the type is float, and the landmark JSON data is defined as follows:

Function Syntax

```
{
        "landmark": {
                "count": 106,
                "data": "xxxx"
        }
}
```

Parameter Description

Parameter	Is it Required	Description
count	Yes	The number of key point coordinates
data	Yes	The key point coordinate base64 data, the coordinate type is float[]

4.1.8 Tracker

Tracker is the face information returned by face tracking in real time, including **Landmark**, **Pose**, and **Rect** information.

The face tracking JSON data is as follows:

Function Syntax

```
{
        "tracker": {
                "blur": 0.0030255913734436035,
                "landmark": {
                        "count": 106,
                        "data": "xxxx"
                },
                "pose": {
                        "pitch": -4.165733814239502,
```

```
                "roll": 0.42814359068870544,
                "yaw": -9.6133241653442383
            },
            "rect": {
                "bottom": 730,
                "left": 529,
                "right": 712,
                "top": 477
            },
            "snapType": "",
            "trackId": 40
        }
}
```


Parameter Description

Parameter	Is it Required	Description
blur	Yes	Face blur degree
landmark	Yes	Face key point coordinates
pose	Yes	Face angle
rect	Yes	Face coordinate point
trackId	Yes	Face tracking ID
snapType	Yes	Reserved value

1. Pose

Pose is to track the face angle in real time, and its JSON data is defined as follows:

Function Syntax


```
{
    "pose": {
        "pitch": -4.165733814239502,
        "roll": 0.42814359068870544,
        "yaw": -9.6133241653442383
    }
```

}

Parameter Description

Parameter	Is it Required	Description
pitch	Yes	Face angle pitch value
roll	Yes	Face angle roll value
yaw	Yes	Face angle yaw value

2. Rect

Rect is the face coordinate point, and its JSON data is defined as follows:

Function Syntax

```
{
        "rect": {
                "bottom": 730,
                "left": 529,
                "right": 712,
                "top": 477
        }
}
```

Parameter Description

Parameter	Is it Required	Description
bottom	Yes	Face bottom coordinates
left	Yes	Face left coordinates
right	Yes	Face right coordinates
top	Yes	Face top coordinates

4.1.9 FaceInfo

The JSON data of FaceInfo is as follows:

Function Syntax

```
{
        "faceInfo": {
                "attribute": {
                        "age": 30,
                        "beauty": -1,
                        "cap": 0,
                        "expression": 0,
                        "eye": -1,
                        "gender": 0,
                        "glasses": -1,
                        "mouth": -1,
                        "mustache": 1,
                        "nation": -1,
                        "respirator": 0,
                        "skinColor": 0,
                        "smile": -1
                },
                "pose": {
                        "pitch": 3.5096845626831055,
                        "roll": -2.404015064239502,
                        "yaw": -13.170290946960449
                },
                "rect": {
                        "bottom": 888,
                        "left": 167,
                        "right": 507,
                        "top": 562
                },
                "landmark": {
                        "count": 106,
                        "data": "xxxx"
                },
                "score": 0.808082
        }
```

}

Parameter Description

Parameter	Is it Required	Description
attribute	Yes	Face attribute
pose	Yes	Face angle
rect	Yes	Face coordinates
landmark	Yes	Face key point coordinates
score	Yes	Face quality

4.1.10 PalmInfo

The JSON data of PalmInfo is as follows:

Function Syntax

```
{
        "palmInfo": {
                "rect": {
                        "x0": 156,
                        "y0": 222,
                        "x1": 356,
                        "y1": 222,
                        "x2": 888,
                        "y2": 167,
                        "x3": 507,
                        "y3": 562
                },
                "imageQuality": 90,
                "templateQuality": 40
        }
}
```

Parameter Description

Parameter	Is it Required	Description
imageQuality	Yes	Palm image quality
templateQuality	Yes	Palm template quality
x0	Yes	The x coordinate of the upper right corner of the

		palm (coordinates are arranged in counterclockwise order)
y0	Yes	The Y coordinate of the upper right corner of the palm (coordinates are arranged in counterclockwise order)
x1	Yes	The x coordinate of the upper left corner of the palm (coordinates are arranged in counterclockwise order)
y1	Yes	The y coordinate of the upper left corner of the palm (coordinates are arranged in counterclockwise order)
x2	Yes	The x coordinate of the lower left corner of the palm (coordinates are arranged in counterclockwise order)
y2	Yes	The y coordinate of the lower left corner of the palm (coordinates are arranged in counterclockwise order)
x3	Yes	The x coordinate of the lower right corner of the palm (coordinates are arranged in counterclockwise order)
y3	Yes	The y coordinate of the lower right corner of the palm (coordinates are arranged in counterclockwise order)

4.1.11 PalmFeature

The JSON data of the palm template is as follows:

Function Syntax

```
{
        "feature": {
                "verTemplate":
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                "verTemplateSize": 26448,
                "preTemplate":
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                "preTemplateSize": 98448
        }
}
```

Parameter Description

Parameter	Is it Required	Description
verTemplate	No	Palm verification/identification template, used for duplication judgment during registration
verTemplateSize	No	Verification/Identification template size

preTemplate	No	Palm pre-registration template, used to merge registration templates
preTemplateSize	No	Palm pre-registration template size

4.2 Face Service-Related Functions

4.2.1 Face Detection

Use HID's **registerFace** to perform face detection on a single photo or directly take a frame from the UVC stream. The requested data is as follows:

Function Syntax

```
{
        "image": {
                "bioType":"face",
                "data":"/9j/4AAQSkZJRgABA",
                "format":"jpeg",
                "width":720,
                "height":1280
        },
        "feature" : "true",
        "faceInfo" : "true",
        "picture" : "true"
}
```

Parameter Description

Parameter	Is it Required	Description
image	No	Image JSON data used for face detection. If this parameter is not included, the module will directly take a frame of image from the UVC stream for face detection by default.
feature	No	Whether to return the face template. If this parameter is not included, the default is false
faceInfo	No	Whether to return face information. If this parameter is not included, the default is false
picture	No	Whether to return the registered face image. If this parameter is not included, the default is false

The JSON data returned during face registration is as follows:

Function Syntax


```
{
        "data": {
                "faces": [{
                        "faceInfo": {
                                "attribute": {
                                        "age": 36,
                                        "beauty": -1,
                                        "cap": 0,
                                        "expression": 0,
                                        "eye": 0,
                                        "gender": 1,
                                        "glasses": -1,
                                        "mouth": -1,
                                        "mustache": 1,
                                        "nation": -1,
                                        "respirator": 0,
                                        "respiratorLevel": 0,
                                        "skinColor": 0,
                                        "smile": -1
                                },
                                "landmark": {
                                        "count": 106,
                                        "data": "xxxxxx"
                                },
                                "pose": {
                                        "pitch": -1.715240478515625,
                                        "roll": -2.3650076389312744,
                                        "yaw": -0.57134813070297241
                                },
                                "rect": {
                                        "bottom": 660,
                                        "left": 220,
                                        "right": 504,
                                        "top": 360
                                },
                                "score": 0.99672424793243408
                        },
                        "feature": {
```



```
                        "bioType": "face",
                        "data": "xxxx",
                        "size": 1024
                },
                "picture": {
                        "bioType": "face",
                        "data": "xxxxxxxxxx",
                        "format": "jpeg",
                        "height": 1280,
                        "width": 720
                }
        }]
    },
    "detail": "success",
    "status": 0
}
```


Parameter Description

Parameter	Is it Required	Description
faces	Yes	One or more face data returned when registering a face
feature	No	The returned face template information. If the registration request is true, it will return, otherwise the information will not be returned.
picture	No	The returned registered face image. If the registration request is true, it will be returned, otherwise the information will not be returned.
faceInfo	No	The returned face information. If the registration request is true, it will return, otherwise the information will not be returned.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
byte[] result = new byte[30 * 1024];
int[] size = new int[1];
int ret = AMTHidManager.instance().registerFace(data.getBytes(), result, size);
if (ret == 0) {
        //success
} else {
        //failed
}
```

4.2.2 Face Recognition and Face Tracking

When the face is visible to the module, the algorithm will track the detected face in real-time and analyze the face attributes. At the same time, the algorithm will also perform live face detection and recognition for better quality.

The face tracking information and face recognition information will be returned through UVC data, and users can get it through [CameraDataCallback]. If the UVC device is not available on the Host side, you can also obtain data through [Polling Recognition Result].

The JSON data for face tracking and recognition is as follows:

Function Syntax

```
{
        "face": [{
                "attribute": {
                        "age": 25,
                        "beauty": -1,
                        "cap": 0,
                        "expression": 0,
                        "eye": -1,
                        "gender": 1,
                        "glasses": -1,
                        "mouth": -1,
                        "mustache": 1,
                        "nation": -1,
                        "respirator": 0,
                        "skinColor": 0,
                        "smile": -1
                },
                "feature": {
                        "data": "xxxxxx",
                        "size": 1024
                },
                "identify": [{
                        "groupId": "",
                        "name": "magic",
```

```
                        "personId": "1180665",
                        "similarity": 0.965038001537323,
                        "userId": "1180665"
                }],
                "liveness": {
                        "irFrameId": 215729,
                        "liveness": 2,
                        "livenessMode": 12,
                        "livenessScore": 0.69999980926513672,
                        "quality": 0.50933307409286499
                },
                "tracker": {
                        "blur": 0.00085997581481933594,
                        "landmark": {
                                "count": 106,
                                "data": "xxxxx"
                        },
                        "pose": {
                                "pitch": -3.7135705947875977,
                                "roll": 3.0353362560272217,
                                "yaw": -23.293550491333008
                        },
                        "rect": {
                                "bottom": 797,
                                "left": 428,
                                "right": 675,
                                "top": 585
                        },
                        "snapType": "",
                        "trackId": 41
                }
        }],
        "label": 1
}
```


Parameter Description

Parameter	Is it Required	Description
attribute	No	Face attribute

feature	No	Face verification template
identify	No	Internal comparison identification information
liveness	No	Face tracking information, including face coordinates, etc.
tracker	No	Face quality
label	Yes	Biological attribute category, 1 is face, 5 is palm

Sample code

```
AMTCameraManager amtCameraManager = new AMTCameraManager();
amtCameraManager.setCallback(new CameraDataCallback() {
        @Override
        public void onFrameDataRecv(VideoData videoData) {
                //image data
        }

        @Override
public void onCustomDataRecv(CustomData customData) {
                //analyze custom data
                //local identify
AMTFaceMatch.dbIdentify(mContext, verTemplate, id, score)
        }
});
```

4.3 Palm Service-Related Functions

4.3.1 Palm Detection

The HID's registerPalm can detect a single 8-bit grayscale photo. If there is no palm photo, this interface will directly take a grayscale image from the UVC near-infrared image stream for palm detection.

The request information for palm detection is as follows:

Function Syntax

```
{
        "image": {
                        "bioType":"palm",
                        "data":"/9j/4AAQSkZJRgABA",
                        "format":"gray",
                        "width":720,
                        "height":1280
        },
        "feature" : "true",
        "palmInfo" : "true",
        "picture" : "true"
}
```

Parameter Description

Parameter	Is it Required	Description
image	No	Detect the image JSON data of the palm. If this parameter is not included, the module will directly take a frame of image from the near-infrared stream for palm detection by default.
feature	No	Whether to return the palm template. If this parameter is not included, the default is false
palmInfo	No	Whether to return palm information. If this parameter is not included, the default is false
picture	No	Whether to return the registered palm image. If this parameter is not included, the default is false

After calling the palm detection interface, the pre-registration template and the verification template will get returned. Here the verification template is for de-duplication judgment, and the pre-registration template is to merge the registration template. When the number of pre-registered

templates reaches 5, it is required to call the merge palm template interface to obtain the merged registration template. And thus, the registration template registers the detected palm.

The JSON data returned by the palm detection is as follows:

Function Syntax


```
{
        "data" : {
                "palms" : [
                {
                "feature" : {
                        "preTemplate" : "xxxxxxx",
                        "preTemplateSize" : 98448,
                        "verTemplate" : "xxxxxx",
                        "verTemplateSize" : 26448
                                                },
                "palmInfo" : {
                        "imageQuality" : 90,
                        "rect" : {
                                "x0" : 585,
                                "x1" : 0,
                                "x2" : 0,
                                "x3" : 499,
                                "y0" : 447,
                                "y1" : 340,
                                "y2" : 818,
                                "y3" : 925
                                                },
                        "templateQuality" : 50
                                        }
                                }
                        ]
                },
                "detail" : "success",
                "status" : 0
        }
```


Parameter Description

Parameter	Is it Required	Description
palms	Yes	One or more palm data returned when registering a palm
feature	No	The returned palm template information. If the registration request is True, it will return, or else there will be no return value.
palmInfo	No	The returned palm template information. If the registration request is True, it will return, or else there will be no return value.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().registerPalm(jsonString.getBytes(), resultByteArray, resultSize);
if (ret == 0) {
                //skip recognized palm
        AMTPalmMatch.dbIdentify(mContext, verTemplate, id, score);
                //merge preTemplate
        AMTHidManager.instance().mergePalmTemplate(jsonData, size);
}else{
                //failed
}
```

4.3.2 Merge Palm Pre-registration Template

Please note, only the merged registration template is possible for registration. Users can call **mergePalm Template** to get the merged registration template.

The returned JSON data format is as follows:

Function Syntax

```
{
        "data": {
                "palm": {
                        "feature": {
                                "mergeTemplate": "xxxxxxxx",
```

```
                "mergeTemplateSize": 8844
            }
        }
    },
    "detail": "success",
    "status": 0
}
```


Parameter Description

Parameter	Is it Required	Description
feature	Yes	The merged palm registration template
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
byte[] jsonData = new byte[200 * 1024];
int[] size = new int[1];
int ret = AMTHidManager.instance().mergePalmTemplate(jsonData, size);
if (ret == 0) {
        //add in algorithm
    AMTPalmMatch.dbAdd(mContext, userPin.getBytes(), regTemplate);
}else{
        //failed
}
```

4.3.3 Palm Recognition

When the palm appears within the range of the module, the algorithm will recognize the current palm. The recognition result is returned through UVC or [Polling Recognition Result].

The returned JSON data of palm recognition is as follows:

Function Syntax

```
{
        "label": 5,
        "palm": [{
                "feature": {
                        "verTemplate": "xxxxxxx",
                        "verTemplateSize": 26448
                },
                "trackInfo": {
                        "imageQuality": 134,
                        "rect": {
                                "x0": 527,
                                "x1": 13,
                                "x2": 10,
                                "x3": 523,
                                "y0": 354,
                                "y1": 349,
                                "y2": 760,
                                "y3": 764
                        }
                }
        }]
}
```

Parameter Description

Parameter	Is it Required	Description
feature	No	Palm template
label	Yes	Bio category, 5 denotes palm
trackInfo	No	Palm trackInfo field, contains palm coordinates and image quality.

Sample code

```
AMTCameraManager amtCameraManager = new AMTCameraManager();
amtCameraManager.setCallback(new CameraDataCallback() {
        @Override
        public void onFrameDataRecv(VideoData videoData) {
                //image data
        }

        @Override
public void onCustomDataRecv(CustomData customData) {
                //analyze custom data
                //local identify
                AMTPalmMatch.dbIdentify(mContext, verTemplate, id, score);
        }
});
```

4.4 Get and Set Configuration Parameters

4.4.1 Common Configuration

The common configuration is mainly used for some basic settings of the module, including attribute analysis, liveness switch, mask recognition, debugging level, etc. Set by calling COMMON_CONFIG of **setConfig**.

The set JSON data is as follows:

Function Syntax

```
{
        "commonSettings": {
                "NIRLiveness": true,
                "VLLiveness": false,
                "attendInterval": 5000,
                "attrInterval": 0,
                "attributeRecog": true,
                "countAlgorithm": false,
                "debugLevel": 0,
```

```
                "drawTrackRect": false,
                "enableStoreAttendLog": true,
                "enableStoreStrangerAttLog": false,
                "faceAEEnabled": true,
                "hacknessThreshold": 0.99000000953674316,
                "infraredPictureFormat": "jpeg",
                "isTrackingMatchMode": true,
                "recogInterval": 1000,
                "recogRespirator": false,
                "recogThreshold": 0.89999997615814209,
                "scoringInterval": 5
        }
}
```


Call **getConfig** to get **COMMON_CONFIG**, and the JSON data obtained is as follows:

Function Syntax


```
{
        "status": 0,
        "detail": "success",
        "data": {
                "commonSettings": {
                        "NIRLiveness": true,
                        "VLLiveness": false,
                        "attendInterval": 5000,
                        "attrInterval": 0,
                        "attributeRecog": true,
                        "countAlgorithm": false,
                        "debugLevel": 0,
                        "drawTrackRect": false,
                        "enableStoreAttendLog": true,
                        "enableStoreStrangerAttLog": false,
                        "faceAEEnabled": true,
                        "hacknessThreshold": 0.99000000953674316,
                        "infraredPictureFormat": "jpeg",
                        "isTrackingMatchMode": true,
                        "recogInterval": 1000,
```



```
                "recogRespirator": false,
                "recogThreshold": 0.89999997615814209,
                "scoringInterval": 5
            }
        }
}
```

Parameter Description

Parameter	Is it Required	Description
NIRLiveness	No	NIR liveness detection
VLLiveness	No	Visible Light liveness detection
attendInterval	No	Matching interval
attrInterval	No	Time interval for detecting face attributes (unit: ms)
attributeRecog	No	Attribute recognition enable switch
countAlgorithm	No	Whether the firmware printing algorithm takes time
debugLevel	No	Algorithm internal module debug level
drawTrackRect	No	UVC image drawing and tracking face frame
enableStoreAttendLog	No	Is enable store matching log
enableStoreStrangerAttLog	No	Is enable store stranger matching log
faceAEEnabled	No	Area exposure enable switch
hacknessThreshold	No	Anti-fake threshold
isTrackingMatchMode	No	True means tracking match mode ;False means detect match mode.
recogInterval	No	Time interval for recognition (unit: ms)
recogRespirator	No	Mask detection enable switch
recogThreshold	No	1:1 verification threshold
scoringInterval	No	Quality detection interval frame
infraredPictureFormat	No	The returned infrared image format. jpeg or gray
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
//set
val json = main.toString().toByteArray()
val size = IntArray(1) { json.size };
AMTHidManager.instance().setConfig(ConfigType.COMMON_CONFIG, json, size)
```

```
//get
val configData = ByteArray(30 * 1024);
val size = IntArray(1)
AMTHidManager.instance().getConfig(ConfigType.COMMON_CONFIG, configData, size)
```

4.4.2 Face Filtering Configuration

The face filtering configuration is primarily for setting various thresholds of the face algorithm. Modifying these configurations will affect the results of face tracking, face attribute analysis, and face recognition.

Set by calling the CAPTURE_FILTER_CONFIG of **setConfig**, and set the JSON data format as follows:

Function Syntax

```
{
        "captureFilter": {
                "blurThreshold": 30,
                "frontThreshold": 50,
                "heightMaxValue": 400,
                "heightMinValue": 40,
                "pitchMaxValue": 30,
                "pitchMinValue": -30,
                "rollMaxValue": 30,
                "rollMinValue": -30,
                "scoreThreshold": 30,
                "widthMaxValue": 400,
                "widthMinValue": 40,
                "yawMaxValue": 30,
                "yawMinValue": -30
        }
}
```

The CAPTURE_FILTER_CONFIG parameter is available through [getConfig], and the obtained JSON data format is as follows:

Function Syntax

```
{
        "status": 0,
        "detail": "success",
        "data": {
                "captureFilter": {
                        "blurThreshold": 30,
                        "frontThreshold": 50,
                        "heightMaxValue": 400,
                        "heightMinValue": 40,
                        "pitchMaxValue": 30,
                        "pitchMinValue": -30,
                        "rollMaxValue": 30,
                        "rollMinValue": -30,
                        "scoreThreshold": 30,
                        "widthMaxValue": 400,
                        "widthMinValue": 40,
                        "yawMaxValue": 30,
                        "yawMinValue": -30
                }
        }
}
```

Parameter Description

Parameter	Is it Required	Description
blurThreshold	No	Image blur degree
frontThreshold	No	Frontal threshold
heightMaxValue	No	Maximum face height threshold
heightMinValue	No	Minimum face height threshold
pitchMaxValue	No	Maximum pitch threshold
pitchMinValue	No	Minimum pitch threshold
rollMaxValue	No	Maximum roll threshold
rollMinValue	No	Minimum roll threshold
scoreThreshold	No	Quality threshold

widthMaxValue	No	Maximum face width threshold
widthMinValue	No	Minimum face width threshold
yawMaxValue	No	Maximum yaw threshold
yawMinValue	No	Minimum yaw threshold
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
        //set
    val json = jsonObject.toString().toByteArray()
    val size = IntArray(1) { json.size }
        AMTHidManager.instance().setConfig(ConfigType.CAPTURE_FILTER_CONFIG, json,
    size)
        //get
val configData = ByteArray(20 * 1024);
val size = IntArray(1)
AMTHidManager.instance().getConfig(ConfigType.CAPTURE_FILTER_CONFIG, configData, size)
```

4.4.3 Motion Detection Configuration

The motion detection configuration controls the sleep mechanism and sensitivity when the device is idle for a long time.

Call **MOTION_DETECT_CONFIG** of **setConfig**, and set the JSON data format as follows:

Function Syntax

```
{
    "MotionDetectionSetting": {
        "brightnessThreshold": 240,
        "idleTimeOutMS": 11000,
        "motionDetectFunOn": true,
        "sensitivityThreshold": 5
    }
}
```

The MOTION_DETECT_CONFIG parameter can be obtained through [getConfig], and the obtained json data format is as follows:

Function Syntax

```
        {
        "status": 0,
        "detail": "success",
        "data": {
                "MotionDetectionSetting": {
                                "brightnessThreshold": 240,
                                "idleTimeOutMS": 11000,
                                "motionDetectFunOn": true,
                                "sensitivityThreshold": 5
                }
        }
}
```

Parameter Description

Parameter	Is it Required	Description
brightnessThreshold	Yes	Brightness threshold [10~1000], indicates the maximum average brightness of the pixel. When this value is exceeded, fill light will not be turned on; otherwise, the fill light will be turned on;
idleTimeOutMS	Yes	This function defines, after how many milliseconds the fill light will be turned off if there is no biometric detection during the idle mode
motionDetectFunOn	Yes	This function defines whether the Motion detection control light function is turned on or not; Value: "true"/"false";
sensitivityThreshold	Yes	This indicates the sensitivity threshold [0 ~ 100], the smaller the value, the more sensitive it is;
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
//set
val json = jsonObject.toString().toByteArray()
val size = IntArray(1) { json.size }
AMTHidManager.instance().setConfig(ConfigType.MOTION_DETECT_CONFIG, json, size)
//get
val data = ByteArray(2048)
val size = IntArray(1)
val ret = AMTHidManager.instance().getConfig(ConfigType.MOTION_DETECT_CONFIG, data, size)
```

4.4.4 Palm Algorithm Configuration

The palm algorithm configuration can control the palm recognition algorithm switch, set the palm recognition threshold, and set the palm image threshold.

By calling PALM_CONFIG of [setConfig], set the JSON data format as follows:

Function Syntax

```
{
        "PALMSetting": {
                "imageQualityThreshold": 60,
                "palmFunOn": true,
                "palmIdentifyThreshold": 576,
                "palmRunState": "match",
                "palmSupportHeight": 1280,
                "palmSupportWidth": 720,
                "templateQualityThreshold": 20
        }
}
```

PALM_CONFIG can be obtained through [getConfig], and the obtained json data format is as follows:

Function Syntax

```
        {
        "status": 0,
        "detail": "success",
        "data": {
                "PALMSetting": {
                        "imageQualityThreshold": 60,
                        "palmFunOn": true,
                        "palmIdentifyThreshold": 576,
                        "palmRunState": "match",
                        "palmSupportHeight": 1280,
                        "palmSupportWidth": 720,
                        "templateQualityThreshold": 20
```

```
                }
            }
}
```

Parameter Description

Parameter	Is it Required	Description
palmFunOn	Yes	Palm function is turned on or not; Value: "true"/"false";
palmIdentifyThreshold	Yes	Palm identification threshold
palmRunState	Yes	Palm algorithm running state, registration state or verification/identification state; Value: "match"/"enroll";
palmSupportWidth	Yes	The image width currently supported by the palm algorithm; (Change is not supported)
palmSupportHeight	Yes	The image height currently supported by the palm algorithm; (Change is not supported)
imageQualityThreshold	Yes	Palm image quality threshold, registration status setting, and verification/identification status may not be set. Default: 60;
templateQualityThreshold	Yes	Palm template quality threshold, registration status setting. Default: 20;
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
//set
val json = jsonObject.toString().toByteArray()
val size = IntArray(1) { json.size }
AMTHidManager.instance().setConfig(ConfigType.PALM_CONFIG, json, size)
//get
val data = ByteArray(20*1024)
val size = IntArray(1)
val ret = AMTHidManager.instance().getConfig(ConfigType.PALM_CONFIG, data, size)
```

4.4.5 Device Information

Device information is the essential information of the module, including firmware version number, HID version number, serial number, etc.

Call the **DEVICE_INFORMATION** of **getConfig**, and the JSON data format of the device information is as follows:

Function Syntax

```
{
        "status": 0,
        "detail": "success",
        "data": {
                "deviceInfo": {
                        "gSDK_VERSION": "0.0.0.33fix4-APP_29-20200825-1658-0-F0001-
gDEP_VERSION=75-gOBJ_VERSION=76",
                        "app": "APP_29",
                        "cpID": "050046ef",
                        "devKey": "02e8e8c90673ef66397740185d4d9020",
                        "sn": "200628-0317",
                        "hidVer": "V1.3.8",
                        "firmVer": "0.3.8UN_33f4-20200826T191308"
                }
        }
}
```

Parameter Description

Parameter	Is it Required	Description
gSDK_VERSION	Yes	Global SDK version
app	Yes	Business process version
cpID	Yes	cpID serial number
devKey	Yes	Device key
sn	Yes	Device serial number
hidVer	Yes	HID service version number
firmVer	Yes	Module firmware version number
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
//get
val data = ByteArray(1024)
val size = IntArray(1)
val ret = AMTHidManager.instance().getConfig(ConfigType.DEVICE_INFORMATION, data, size)
```

4.4.6 Device Time Configuration

Developers can synchronize the device time by calling DEVICE_TIME of [AMTHidManager_setConfig].

Set the JSON data format as follows:

Function Syntax

```
{
        "syncTime":"2020-11-23 16:44:52"
}
```

DEVICE_TIME can be obtained through [AMTHidManager_getConfig], and the obtained JSON data format is as follows:

Function Syntax

```
{
        "data" : {
                        "sysTime" : "2020-11-23 16:44:54"
                                },
                "detail" : "success",
                "status" : 0
        }
```

Parameter Description

Parameter	Is it Required	Description
sysTime	Yes	Device time
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
//get
val result = ByteArray(256)
val size = IntArray(1)
AMTHidManager.instance().getConfig(ConfigType.DEVICE_TIME, result, size)
//set
AMTHidManager.instance().setConfig(ConfigType.DEVICE_TIME, data, size)
```

4.5 Operation Related

4.5.1 Snapshot

Through the HID command, you can capture a frame of the current image and return it to the host.

The returned JSON data is as follows:

Function Syntax

```
{
        "data": {
                "snapshot": {
                        "data": "xxxxxxxxxx",
                        "frameId": 163847,
                        "height": 1280,
                        "timeStamp": 6656928,
                        "type": "jpeg",
                        "width": 720
                }
        },
        "detail": "success",
        "status": 0
}
```

Parameter Description

Parameter	Is it Required	Description
frameId	Yes	Frame index
height	Yes	Image height
width	Yes	Image width

data	Yes	Image data encrypted by Base64. The format of the visible light image is JPEG. And the format of NIR image may be JPEG or GRAY, depending on the parameter infraredPictureFormat in commonSettings.
timeStamp	Yes	Timestamp
type	Yes	Image data type
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
byte[] snapData = new byte[2 * 1024 * 1024];
int[] length = new int[1];
int ret = AMTHidManager.instance().snapShot(SnapType.SNAP_RGB, snapData, length);
//or
int ret = AMTHidManager.instance().snapShot(SnapType.SNAP_GRAY, snapData, length);
```

4.6 Module Data Management

There is a set of data management mechanism inside the module, which can manage the data (users, biological templates, matching records, etc.) inside the module through the [manageModuleData] of the HID communication interface.

4.6.1 Add User

Add a user to the module through [manageModuleData].

The JSON data format for adding users is as follows:

Function Syntax

```
{
        "personId": "12345",
        "groupId": "DA640D7CE7544B33A3A1C81CD95D3E66 ",
        "userId": "12345",
        "name": "Rio",
        "age": 30,
        "gender": "male",
        "updateTime": "20200331123025",
        "image": [{
                        "bioType": "face",
                        "data": "/9j/4AAQSkZJRgABA",
                        "format": "jpeg",
```

```
                        "width": 720,
                        "height": 1280
                },
                {
                        "bioType": "palm",
                        "data": "/9j/4AAQSkZJRgABA",
                        "format": "gray",
                        "width": 720,
                        "height": 1280
                }
        ],
        "features": [{
                        "bioType": "face",
                        "data":
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                        "size": 1024
                },
                {
                        "bioType": "palm",
                        "data":
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                        "size": 1024
                }
        ]
}
```


Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which must be consistent with the userId
groupId	No	User group Id
userId	Yes	The unique ID of the user, which must be consistent with the personId
name	Yes	User name
gender	Yes	User gender
age	No	User age
updateTime	No	Update time
image	No	Register a photo of a face or palm.
features	No	Register a biometric template for the face or palm.

Response for Add user:

```
{
        "status": 0,
        "detail": "success",
        "data": {
                        "personId": "12345"
                }
        }
```

Sample code:

```
byte[] resultByteArray = new byte[20 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.ADD_PERSON, null, resultByteArray,
resultSize);
if (i == 0) {
        //success
} else {
         //failed
}
```

4.6.2 Delete User

This function deletes a specified user from the module through DEL_PERSON of [manageModuleData].

The JSON data format of the deleted user is as follows:

Function Syntax

```
{
        "personId": "570"
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which is consistent with the userId

The format of the JSON data returned by the deleted user is as follows:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.DEL_PERSON, json.getBytes(),
resultByteArray, resultSize);
if (i == 0) {
        //success
} else {
         //failed
}
```

4.6.3 Clear Users

Clear all users in the module through the CLEAR command of [manageModuleData].

The JSON data format of the empty user response is as follows:

Function Syntax

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.CLEAR, null, resultByteArray,
resultSize);
```

```
if (i == 0) {
        //success
} else {
        //failed
}
```

4.6.4 Query User

Through GET_PERSON of [manageModuleData], the developer can query all the information of a specified user that already exists in the module.

The JSON data format of the query user is as follows:

Function Syntax

```
{
        "personId": "123456",
        "data": [{
                        "bioType": "face",
                        "feature": true,
                        "picture": true
                },
                {
                        "bioType": "palm",
                        "feature": true
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which is consistent with the userId
bioType	No	Type, can be "face" or "palm"
feature	No	Whether to return to the template. Note that a person can have multiple face and palm templates, so multiple templates may be returned. Please pay attention to allocating appropriate buffers.
picture	No	Whether to return the registered photo. Only valid when the biotype is a face, the palm does not save the registered photo.

The format of the JSON data returned by the query user is as follows:

Function Syntax

```
{
        "data": {
                "age": -1,
                "features": [{
                                "bioType": "face",
                                "data": "xxxxx",
                                "size": 1024
                        },
                        {
                                "bioType": "face",
                                "data": "xxxxxx",
                                "size": 1024
                        },
                        {
                                "bioType": "palm",
                                "data": "xxxxxxx",
                                "size": 8844
                        },
                        {
                                "bioType": "palm",
                                "data": "xxxxxxx",
                                "size": 8844
                        }
                ],
                "gender": "male",
                "groupId": "",
                "images": [{
                                "bioType": "face",
                                "data": "",
                                "format": "jpeg",
                                "height": 983,
                                "width": 612
                        },
                        {
```

```
                                "bioType": "face",
                                "data": "",
                                "format": "jpeg",
                                "height": 800,
                                "width": 578
                        }
                ],
                "name": "1180665",
                "personId": "1180665",
                "updateTime": "20201102175244",
                "userId": "1180665"
        },
        "detail": "success",
        "status": 0
}
```


Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which must be consistent with the userId
groupId	No	User group Id
userId	Yes	The unique ID of the user, which must be consistent with the personId
name	Yes	User name
gender	Yes	User gender
age	No	User age
updateTime	No	Update time
images	No	Register a photo of a face or palm.
features	No	Register a biometric template for the face or palm.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.GET_PERSON, param, resultByteArray,
resultSize);
if (i == 0) {
        //success
```

```
} else {
        //failed
}
```

4.6.5 Query All Users

Through QUERY_ALL_PERSON of [manageModuleData], you can query all user information in the module.

The JSON data format for querying all user information is as follows:

Function Syntax

```
{
        "pageIndex" : 1,
        "pageSize" : 20
}
```

Parameter Description

Parameter	Is it Required	Description
pageIndex	Yes	Page number
pageSize	Yes	Page size, if the page size is 0, query all

The JSON data format returned by querying all users is as follows:

Function Syntax

```
{
        "data": [{
                        "age": -1,
                        "face": 2,
                        "gender": "male",
                        "groupId": "",
                        "name": "2855N",
                        "palm": 1,
                        "personId": "2855",
                        "phone": "",
                        "updateTime": "20201016191450",
                        "userId": "2855"
                }
```

```
        ],
        "detail": "success",
        "status": 0
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which must be consistent with the userId
groupId	No	User group Id
userId	Yes	The unique ID of the user, which must be consistent with the personId
name	Yes	User name
age	No	User age
updateTime	No	Update time
gender	No	User gender
face	No	Number of faces registered by the user
palm	No	Number of palms registered by the user
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

Function Syntax

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.QUERY_ALL_PERSON, null, resultByteArray, resultSize);
if (i == 0) {
        //success
} else {
        //failed
}
```

4.6.6 Get Person Statistics

Through the QUERY_STATISTICS of [manageModuleData], the user information existing in the current module can be counted and returned.

The JSON data format for statistical user information is as follows:

Function Syntax

```
{
        "data" : {
                        "databaseSize" : 40960,
                        "faceCount" : 0,
                        "featureCount" : 0,
                        "featureSize" : 16,
                        "groupCount" : 0,
                        "palmCount" : 0,
                        "palmFeatureCount" : 0,
                        "palmFeatureSize" : 16,
                        "personCount" : 1,
                        "pictureCount" : 1,
                        "pictureSize" : 16
                                },
                "detail" : "success",
                "status" : 0
        }
```

Parameter Description

Parameter	Is it Required	Description
databaseSize	Yes	Total database size, in bytes
faceCount	Yes	The total number of face records in the face table in the database
featureCount	Yes	Total number of face templates in memory, calculated according to userId
featureSize	Yes	The total size of the face template, in bytes
groupCount	Yes	Total number of table records in the database
palmCount	Yes	The total number of palm records in the database palm table
palmFeatureCount	Yes	Total number of palm templates in memory
palmFeatureSize	Yes	The size of the space occupied by the palm template storage file
personCount	Yes	The total number of personnel records in the database personnel table
pictureCount	Yes	Number of registered photos

pictureSize	Yes	The total size of the picture, in bytes
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.QUERY_STATISTICS, null,
resultByteArray, resultSize);
if (i == 0) {
        //success
} else {
        //failed
}
```

4.6.7 Polling Recognition Result

The polling recognition result is primarily for devices that do not support the UVC protocol. The Host can obtain the face and palm recognition results through polling. The device caches up to 8 recognition results, and the last one in the array is the latest recognition result.

Polling is carried out through [pollMatchResult], and the face JSON data format returned during polling is as follows:

Function Syntax

```
{
        "events": [{
                "label": 1,
                "face": [{
                        "identify": [{
                                "groupId": "",
                                "name": "3123",
                                "personId": "q123",
                                "similarity": 0.9328765869140625,
                                "userId": "q123"
                        }]
                }]
        }]
}
```

The palm json data format of the reply during polling is as follows:

Function Syntax

```
{
        "events": [{
                "label": 5,
                "palm": [{
                        "identify": [{
                                "groupId": "",
                                "name": "3123",
                                "personId": "q123",
                                "similarity": 705,
                                "userId": "q123"
                        }]
                }]
        }]
}
```

Parameter Description

Parameter	Is it Required	Description
label	Yes	Biological attribute category, 1 is face, 5 is palm
identify	Yes	Identification information
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code

```
byte[] resultArray = new byte[20 * 1024];
int[] resultLength = new int[1];
int ret = AMTHidManager.instance().pollMatchResult(resultArray, resultLength);
if (ret == 0) {
    //success
}else{
    //failed
}
```

4.6.8 Face Registration

The Face Registration for the users in the module is processed through local photos or registered face templates. If there an image and a template at the same time, the photo is preferred for registration.

1. Add Faces through Photos

Developers can add one or more face images to the designated users through local photos.

The JSON data for adding face is as follows:

Function Syntax

```
{
        "personId": "12345",
        "image": [
                {
                        "bioType":"face",
                        "data":"/9j/4AAQSkZJRgABA",
                        "format":"jpeg",
                        "width":720,
                        "height":1280
                },
                {
                        "bioType":"face",
                        "data":"/9j/4AAQSkZJRgABA",
                        "format":"jpeg",
                        "width":720,
                        "height":1280
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the face
image	No	Face photos used to register faces

Add face Response:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.ADD_FACE, null, resultByteArray,
resultSize);
if (i == 0) {
        //success
} else {
         //failed
}
```

2. Add Faces through Templates

The face is added to the designated users in the module through face templates that have been registered locally or face templates registered on other devices.

Add face request JSON data as follows:

Function Syntax

```
{
        "personId": "12345",
        "features": [
                {
                        "bioType":"face",
                        "data" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                        "size" : 1024
                },
                {
                        "bioType":"face",
                        "data" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                        "size" : 1024
```

```
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the face
features	No	Face templates used to register faces

Add face reply:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.ADD_FACE, null, resultByteArray, resultSize);
if (i == 0) {
        //success
} else {
        //failed
}
```

4.6.9 Palm Registration

The Palm Registration for the users is processed through local photos or registered palm templates. If there an image and a template at the same time, the photo is preferred for registration.

1. Add Palms through Photos

The palm images can be added to designated users through local photos.

The JSON data for adding palm is as follows:

Function Syntax

```
{
        "personId": "12345",
        "images": [
                {
                        "bioType":"palm",
                        "data":"/9j/4AAQSkZJRgABA",
                        "format":"gray",
                        "width":720,
                        "height":1280
                },
                {
                        "bioType":"palm",
                        "data":"/9j/4AAQSkZJRgABA",
                        "format":"gray",
                        "width":720,
                        "height":1280
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the palm
images	No	8-bit grayscale image for registering the palm

Add palm reply:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.ADD_PALM, null, resultByteArray,
resultSize);
if (i == 0) {
        //success
} else {
         //failed
}
```

2. Add Palms through Templates

Users can add palms to designated users in the module through the palm templates that have been registered locally or the palm templates registered on other devices.

The JSON data for the Add palm request is as follows:

Function Syntax

```
{
        "personId": "12345",
        "features": [
                {
                        "bioType":"palm",
                        "data" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                        "size" : 1024
                },
                {
                        "bioType":"palm",
                        "data" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
```

```
                "size" : 1024
            }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the palm
features	No	Palm templates used to register palms

Add palm reply:

```
{
    "status": 0,
    "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[1024];
int[] resultSize = new int[]{resultByteArray.length};
int i = AMTHidManager.instance().manageUser(ManageType.ADD_PALM, null, resultByteArray, resultSize);
if (i == 0) {
    //success
} else {
     //failed
}
```

4.6.10 Export Matching Log Record

Through EXPORT_ATT_RECORD of [AMTHidManager_manageModuledata], the developer can export all matching records in the module.

The JSON data format for exporting matching records is as follows:

Function Syntax

```
    {
```

```
        "startId" : 1,
        "reqCount" : 20,
        "needImg" : false
}
```

Parameter Description

Parameter	Is it Required	Description
startId	Yes	Start id of matching record
reqCount	Yes	Request count of matching record
needImg	Yes	Whether to export ~~attendance~~ photos. The module does not store ~~attendance~~ photo as default. Please enable enableStoreAttendPhoto in commonSettings once needed.

The JSON data format returned by exporting matching record is as follows:

Function Syntax

```
{
        "data" : [
                {
                        "authType" : "face",
                        "deviceid" : "05004700",
                        "groupId" : "",
                        "id" : 1,
                        "name" : "kobe",
                        "personId" : "kobe",
                        "respirator" : 0,
                        "timestamp" : "2020-12-10 15:55:28.677",
                        "userid" : "kobe"
                }
            ],
        "detail" : "success",
        "recordCount" : 1,
        "status" : 0
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique ID of the user, which must be consistent with the userId
groupId	Yes	User group Id
deviceid	Yes	The value of cpuid info in DEVICE_INFORMATION
userid	Yes	The unique ID of the user, which must be consistent with the personId
name	Yes	User name
authType	Yes	Face or palm

respirator	Yes	With mask or without mask
timestamp	Yes	Check-in time
id	Yes	Index of matching log
recordCount	Yes	Matching record count
status	Yes	Data reply status value, where 0 means success. Please refer to [Appendix 4] for others.

Sample code:

```
byte[] result = new byte[1024 * 1024 * 3];
int[] length = new int[]{result.length};
int ret = AMTHidManager.instance().manageModuleData(ManageType.QUERY_ATT_RECORD,
                        jsonString.getBytes(), result, length);
```

4.6.11 Clear Matching Log Record

Through CLEAR_ATT_RECORD of [AMTHidManager_manageModuledata], the developer can clear all the matching record in the module.

The JSON data format returned by clearing all matching record is as follows:

Function Syntax

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] result = new byte[4096];
int[] length = new int[]{result.length};
int i = AMTHidManager.instance().manageModuleData(ManageType.CLEAR_ATT_RECORD,
                        null, result, length);
```

4.6.12 Cache Registration

To solve the problem of the poor performance of the device, insufficient memory for UVC preview or the lack of image processing capabilities, a data caching mechanism is designed inside the module to assist in the face and palm registration.

- After entering the registration mode, the module will stop the verification mode and create a maximum of 32 cache space. This cache space stores the data related to face and palm registration.
- After exiting the registration mode, the module will restore the verification mode and clear all data in the cache space.

1. Enter Registration Mode

To enter the registration mode without additional parameters, send the Reg Start command.

The format of the returned JSON data is as follows:

Function Syntax

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.REG_START, null,resultByteArray, resultSize);
if (ret == 0) {
        //success
}else{
        //failed
}
```

The format of the returned json data is as follows:

```
{
        "status": 0,
        "detail": "success"
}
```

2. Exit Registration Mode

To exit the registration mode without additional parameters, send the Reg End command.

The format of the returned JSON data is as follows:

Function Syntax

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.REG_END, null,resultByteArray, resultSize);
if (ret == 0) {
        //success
}else{
        //failed
}
```

The format of the returned JSON data is as follows:

```
{
        "status": 0,
        "detail": "success"
}
```

3. Detect Face

The user can use the face photo or the video stream in the module, in the cache registration mode, by sending the DETECT_FACE_REG command to obtain the Cache Id corresponding to the face and the required information.

The requested JSON data is as follows:

Function Syntax

```
{
        "image": {
                "bioType":"face",
                "data":"/9j/4AAQSkZJRgABA",
                "format":"jpeg",
```

```
                "width":720,
                "height":1280
        },
        "feature" : "true",
        "faceInfo" : "true",
        "picture" : "true"
}
```

Parameter Description

Parameter	Is it Required	Description
image	No	Image of registered face. If this parameter is empty, the module will directly take a frame of image from the stream for registration by default.
feature	No	Whether to return the face template. If this parameter is not included, the default is false
faceInfo	No	Whether to return face information. If this parameter is not included, the default is false
picture	No	Whether to return the registered face image. If this parameter is not included, the default is false

The format of the returned JSON data is as follows:

Function Syntax

```
{
        "status": 0,
        "detail": "success",
        "data": {
                "faces":[
                        {
                                "faceInfo" : {
                                "attribute" : {
                                "age" : 30,
                                "beauty" : -1,
                                "cap" : 0,
                                "expression" : 0,
                                "eye" : -1,
                                "gender" : 0,
                                "glasses" : -1,
                                "mouth" : -1,
                                "mustache" : 1,
                                "nation" : -1,
```

```
                                "respirator" : 0,
                                "skinColor" : 0,
                                "smile" : -1
                                },
                                "pose" : {
                                        "pitch" : 3.5096845626831055,
                                        "roll" : -2.404015064239502,
                                        "yaw" : -13.170290946960449
                                },
                                "rect" : {
                                        "bottom" : 888,
                                        "left" : 167,
                                        "right" : 507,
                                        "top" : 562
                                        },
                                "landmark" : {
                                "count" : 106,
                                "data" :
"EAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGwiT0lyPS7/ZYEvf2WBL0CMCO9/pW3vAP9"
                                },
                                "score": 0.808082
                                },
                                "feature" : {
                                        "bioType":"face",
                                        "data" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                                "size" : 1024
                        },
                                "picture": {
                                        "bioType":"face",
                                        "data":"/9j/4AAQSkZJRgABA",
                                        "format":"jpeg",
                                        "width":720,
                                        "height":1280
                                },
                                "cacheId":{
                                        "bioType":"face",
                                        "data":1
                                }
                                }
                        ]
                }
        }
```


Parameter Description

Parameter	Is it Required	Description
faces	Yes	One or more face data returned when registering a face
feature	No	The returned face template information. If the registration request is true, it will return, otherwise the information will not be returned.
picture	No	The returned registered face image. If the registration request is true, it will be returned, otherwise the information will not be returned.
faceInfo	No	The returned face information. If the registration request is true, it will return, otherwise the information will not be returned.
cacheId	No	The returned face cache information Id.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.DETECT_FACE_REG,
jsonString.getBytes(),
                resultByteArray, resultSize);
if (ret == 0) {
        //success
}else{
        //failed
}
```

4. Add Registered Face

By adding the face **CacheID** generated by detecting the face in the cache mode the detected face will get added to the specified user in the module.

The JSON data format for adding faces is as follows:

Function Syntax

```
{
        "personId": "12345",
        "cacheIds":[
                {
                        "bioType":"face",
                        "data":1
                },
                {
                        "bioType":"face",
                        "data":2
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the face
cacheIds	No	The internal cache ID of the face used to register the face.

The format of the JSON data returned by adding a face is as follows:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[20 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.ADD_FACE_REG,
jsonString.getBytes(),resultByteArray, resultSize);
```

```
if (ret == 0) {
        //success
} else {
        //failed
}
```

5. Detect Palm

The user can use the palm photo or the video stream in the module to send the DETECT_PALM_REG command in the cache registration mode to obtain the corresponding palm pre-registration template Cache Id and required palm information.

The requested JSON data is as follows:

Function Syntax

```
{
        "image": {
                "bioType":"palm",
                "data":"/9j/4AAQSkZJRgABA",
                "format":"gray",
                "width":720,
                "height":1280
        },
        "feature" : "true",
        "palmInfo" : "true",
        "picture" : "true"
}
```

Parameter Description

Parameter	Is it Required	Description
image	No	Register a picture of the palm. If this parameter is empty, the module will directly take a frame of image from the stream for registration by default.
feature	No	Whether to return the palm template. If this parameter is not included, the default is false
palmInfo	No	Whether to return palm information. If this parameter is not included, the default is false
picture	No	Whether to return the registered palm image. If this parameter is not included, the default is false

The format of the returned JSON data is as follows:

Function Syntax


```
{
        "status": 0,
        "detail": "success",
        "data": {
                "palms":[
                        {
                        "palmInfo" : {
                                "rect" : {
                                        "x0" : 156,
                                        "y0" : 222,
                                        "x1" : 356,
                                        "y1" : 222,
                                        "x2" : 888,
                                        "y2" : 167,
                                        "x3" : 507,
                                        "y3" : 562
                                        },
                                "imageQuality": 90,
                                "templateQuality": 40
                        },
                        "feature" : {
                                "verTemplate" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                                "verTemplateSize" : 26448,
                                "preTemplate" :
"5C+ju39I273/lGq9gLAJvv+WhL39lgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/loS9fq7vPYGw",
                                "preTemplateSize" : 98448
                                        },
                        "picture" : {
                                "bioType":"palm",
                                "data":"/9j/4AAQSkZJRgABA",
                                "format":"gray",
                                "width":720,
                                "height":1280
                                },
                        "cacheId":
                                {
                                "bioType":"palm",
                                "data":[1]
                                }
```



```
                    }
                ]
            }
}
```

Parameter Description

Parameter	Is it Required	Description
palms	Yes	One or more palm data returned when registering a palm
feature	No	The returned palm template information. If the registration request is true, it will return, otherwise the information will not be returned.
picture	No	The returned registered palm image. If the registration request is true, it will be returned, otherwise the information will not be returned.
palmInfo	No	The returned palm information. If the registration request is true, it will return, otherwise the information will not be returned.
cacheId	No	The returned palm cache information Id.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.DETECT_PALM_REG,
jsonString.getBytes(),resultByteArray, resultSize);
if (ret == 0) {
        //success
}else{
        //failed
}
```

6. Merge Cached Palm Pre-registration Template

When the CacheId pre-registration template obtained by palm detection is 5, the merged registration template is acquired by sending the CacheId pre-registration template via the MERGE_PALM_REG command.

The JSON data requested to be combined as follows:

Function Syntax

```
{
        "feature" : true,
        "cacheIds":[
                {
                        "bioType":"palm",
                        "data":1
                },
                {
                        "bioType":"palm",
                        "data":2
                },
                {
                        "bioType":"palm",
                        "data":3
                },
                {
                        "bioType":"palm",
                        "data":4
                },
                {
                        "bioType":"palm",
                        "data":5
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
feature	Yes	Whether to return the merged palm registration template. If this parameter is not included, the default is false
cacheIds	yes	The CacheId of the pre-registered template must be 5. CacheId data

Response

Function Syntax

```
{
        "status": 0,
        "detail": "success",
        "data":{
                "palm":{
                        "feature":{
                                "mergeTemplate" : "xxx",
                                "mergeTemplateSize" : 8844
                        },
                        "cacheId":{
                                "bioType":"palm",
                                "data":1
                        }
                }
        }
}
```

Parameter Description

Parameter	Is it Required	Description
palm	Yes	Returned palm details.
feature	yes	The returned merged palm template
cacheId	yes	The returned CacheId of the merged palm template.
status	Yes	Data reply status value. 0 means success, please refer to [Appendix 4] for others.

Sample code:

```
byte[] resultByteArray = new byte[1024 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.MERGE_PALM_REG,
jsonString.getBytes(),resultByteArray, resultSize);
if (ret == 0) {
        //success
}else{
        //failed
}
```

7. Add Registered Palm

By merging the palm registration template CacheID generated after caching the palms, you can add palms to designated users in the module.

The JSON data format of the added palm is as follows:

Function Syntax

```
{
        "personId": "12345",
        "cacheIds":[
                {
                        "bioType":"palm",
                        "data":1
                }
        ]
}
```

Parameter Description

Parameter	Is it Required	Description
personId	Yes	The unique personId of the user who registered the palm
cacheIds	No	The merged registration template cache Id used to register the palm

Add palm Response:

```
{
        "status": 0,
        "detail": "success"
}
```

Sample code:

```
byte[] resultByteArray = new byte[20 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
int ret = AMTHidManager.instance().manageUser(ManageType.ADD_PALM_REG,
jsonString.getBytes(),resultByteArray, resultSize);
if (ret == 0) {
        //success
} else {
        //failed
```

```
}
```

8. Delete Cached Data

Due to the limited memory space of the module, the cache registration mode has a maximum of 32 storage spaces for caching. And if the user needs to remove the cached data that does not meet the requirements can be deleted to free up more space for filtering.

The JSON data format for deleting cached data is as follows:

Function Syntax

```
{
        "cacheId":{
                "bioType":"face",
                "data":1
        }
}
```

Parameter Description

Parameter	Is it Required	Description
bioType	Yes	Cache data type, face, or palm
data	Yes	Unique ID of cached data

Sample code:

```
byte[] resultByteArray = new byte[20 * 1024];
int[] resultSize = new int[]{resultByteArray.length};
//delete face cache id
int ret = AMTHidManager.instance().manageUser(ManageType.DEL_FACE_CACHE_ID,
jsonString.getBytes(),resultByteArray, resultSize);
//delete palm cache id
int ret = AMTHidManager.instance().manageUser(ManageType.DEL_PALM_CACHE_ID,
jsonString.getBytes(),resultByteArray, resultSize);
if (ret == 0) {
        //success
} else {
        //failed
}
```

5 Appendix

Appendix 1: HID Communication Interface Error Code

Error Code	Description
0	Success
-1	Failed to turn on the device
-2	The device is not turned on
-3	Parameter error
-4	Memory allocation failed; not enough memory allocated
-5	The memory space requested by the user is not enough
-6	Failed to send command
-7	Failed to read data, timeout
-8	Setting failed
-9	Failed to open the file
-10	Abnormal file
-11	Unauthorized protocol header
-12	Sum check failed
-13	File reading failed
-14	Data return length is incorrect
-15	Data return status is incorrect
-16	Data return is empty
-17	Null pointer exception
-18	USB device without permission
-19	USB device not found
-20	USB device failed to open

Appendix 2: Local Face Verification Interface Error Code

Error Code	Description
0	Success
-1	Parameter error
-2	Failed to request for memory allocation
-3	Not enough memory
-4	Unauthorized ID
-5	Database is empty
-6	Duplicate ID

Appendix 3: Local Palm Verification Interface Error Code

Error Code	Description
0	Success
-1	Parameter error
-2	Failed to request for memory allocation
-3	Not enough memory
-4	Unsupported Interface
-5	Failed to load the algorithm library
-6	Failed to initialize the algorithm library
-103	Unauthorized ID
-106	Duplicate ID
-200	Database is full

Appendix 4: Data Communication Error Code

Error Code	Description	Additional Remark
0	Success	
100	System needs to restart	To take effect of the changes made to the configuration parameter requires a restart.
404	Illegal resource request	The requested URL address is incorrect, or the requested resource is not available
405	Invalid request field value	A field value exceeds the valid value range; for example, the CameraId has a value other than 0,1
406	Required field parsing failed	When parsing the data, the required field not found (Required field)
407	Failed to find file in MAP table	When calling the import and export series interface, the passed file name is not found in the file list.
409	The length of the request message body is invalid	Generally, the setting interface is called, but the length of the message body is 0.
410	Failed to deserialize the message body	Failed to deserialize the request message body.
411	Invalid configuration parameters	The configuration parameter is not in the valid range and is returned by calling the setting interface.
412	Base64 data decoding failed	Exception when decoding base64 picture data
413	Picture data is incomplete	The format of the base64 encoded image is incorrect.
415	Incomplete batch file	When calling the batch import and export interface, some files are not imported or exported, and the integrity check fails
420	MD5 verification failed	Failed to verify the MD5 value of the file
421	Token verification failed	The Token value is inconsistent with the Token preset by the system
502	Service busy	The system is processing other requests.
503	Not enough storage space	Not enough storage space.
504	The server failed to allocate memory	Error when the system dynamically allocates memory.
505	The server failed to open the file	Failed to open file.
506	Failed to create file directory	mkdir directory failed
507	The server failed to read the file	Failed to read the file

508	Failed to write file on server	Failed to write the file
509	File does not exist	Files that exist logically do not actually exist.
510	Failed to traverse the file directory	Failed to traverse the file list.
511	Failed to open database file	Generally, the database file is invalid or incomplete.
514	Failed to load feature vector	Generally, the feature of the vector file is invalid or incomplete.
515	Failed to calculate MD5 value	Failed to calculate MD5 value of the file.
520	Failed to find configuration file	Generally, when obtaining or setting the configuration file, the corresponding configuration parameter is not found
521	Failed to find service	Cannot find a valid StackService; generally, there is a problem with the client passing value.
522	Unsupported service	Generally, an error will get reported when calling the interface related to face recognition to the capture camera, such as the information of the registered person.
523	Database is not started	Database is not loaded
524	The database is not ready	The database is being exported or imported, or the project is being loaded.
525	Database query failed	Failed to call SQL query interface.
526	Database insert failed	Failed to call SQL insert interface.
527	Database update failed	Failed to call SQL update interface.
528	Database deletion failed	Failed to call SQL delete interface.
529	Duplicate database UUID	There are entries with the same UUID when querying the database.
530	JPEG image to YUV failed	The resolution may exceed 1280 x 720, or it may not be a JPEG image.
531	No valid information found	The database search table did not find the data entry that meets the query conditions.
532	Feature matching failed	The matching score is less than the specified threshold.
533	Failed to wait for a valid frame to capture	When calling the capture interface, the valid frame data read within the timeout period.
535	Color gamut conversion failed	Failed to convert YUV to RGB in the algorithm.

536	The image format does not meet the requirements	The image format passed to the algorithm does not meet the requirements
537	Algorithm handle is empty	The corresponding algorithm is not initialized, or the initialization fails, resulting in an empty call handle.
538	Failed to call the face quality analysis interface	It may be related to the incoming face Landmark parameter.
540	DSP is not enabled	The DSP corresponding to the algorithm is not started.
541	Face detection call failed	Failed to call the face detection API.
542	No face detected	Calling the face detection API does not detect the face, generally because the picture's face is not clear enough.
543	Face feature extraction failed	Generally, the face is not clear enough.
544	Does not support attribute analysis	The attribute analysis interface is not initialized, or the initialization fails.
545	Face alignment call failed	Failed to call the face alignment interface, which may be related to face clarity.
546	Face attribute analysis failed	Failed to call the face attribute analysis interface, which may be related to the clarity of the face.
547	Live body detection failed	The call of the live body detection interface failed due to an unknown error.
548	Duplicate user ID	Please check if the user ID has been repeated.
551	Live body detection failed	Failed to call the live body detection interface caused by the face angle.
552	Live body detection failed	Failed to call the live body detection interface caused by the blurred face.
553	Live body detection failed	Failed to call the live body detection interface caused by face size.

190 Bluegrass Valley Pkwy,

Alpharetta, GA 30005, USA

E-mail: info@armatura.us

www.armatura.us