API Development Manual:

AMTMultiBio SDK For Windows

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 Windows**.

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 a standard Win32 APIs that support C, C++, C# language development. It can work on both 32-bit and 64-bit operating system of Windows XP SP3 or higher.

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.

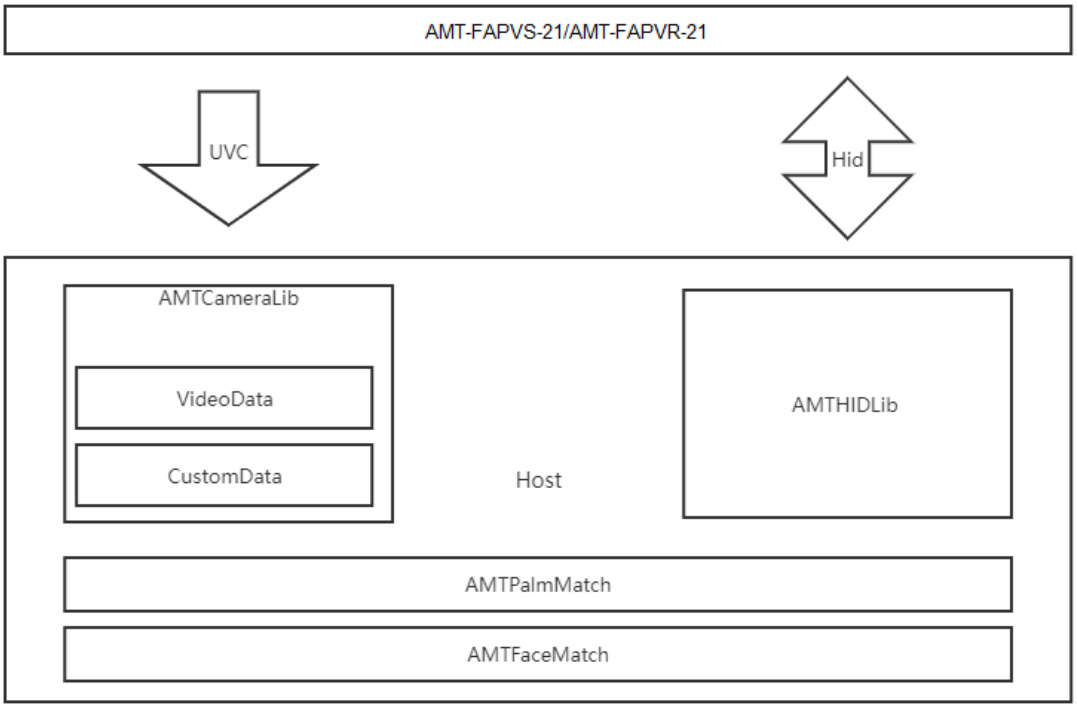
The Match-on-Host mode can transmit images and video data of face and palm through UVC at high speed. Also, it can use the advantages of the high-performance host, which quickly realizes face and palm recognition.

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, then the SDK architecture is mainly composed of **AMTCameraLib**, **AMTHIDLib**, **AMTFaceMatch**, and **AMTPalmMatch**. The overall structure of the SDK displayed in the figure.

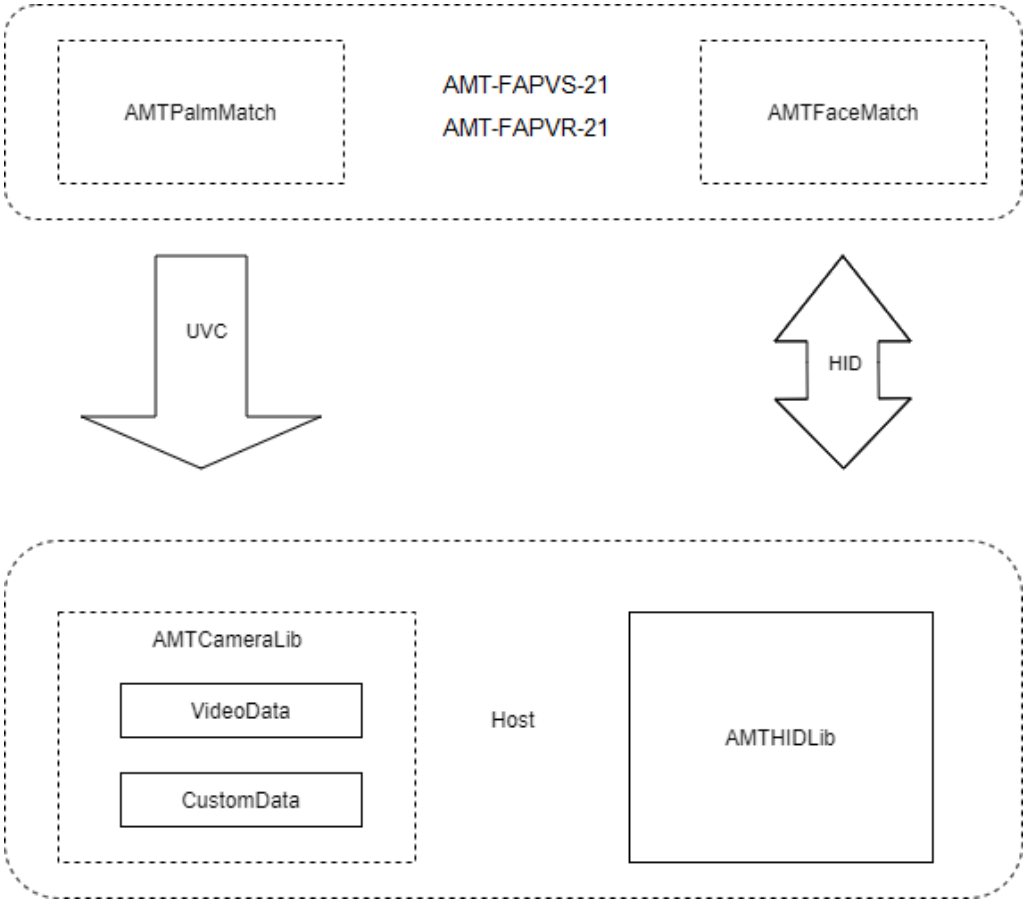


Process Description

- AMTCameraLib acquires UVC video streams and custom data (including biometric information such as faces, palms, etc.).
- AMTHIDLib loads and sets parameters, registers, upgrades, and so on.
- AMTPalmMatch is the algorithm library for palm recognition.
- AMTFaceMatch is the algorithm library for face recognition.

2.1.2 Match-on-Module Mode

If the user uses the Match-on-Module mode, the SDK architecture is mainly composed of **AMTCameraLib** and **AMTHIDLib**. The overall structure of the SDK displayed in the figure below:



Process Description

- AMTCameraLib acquires UVC video streams and custom data including, biometric information and recognition information such as faces, palms and more.
- AMTHIDLib loads and sets parameters, upgrades, manage data and more.
- The Match-on-Module Mode cannot only transmit face and palm data through UVC but also obtains the data of face and palm 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

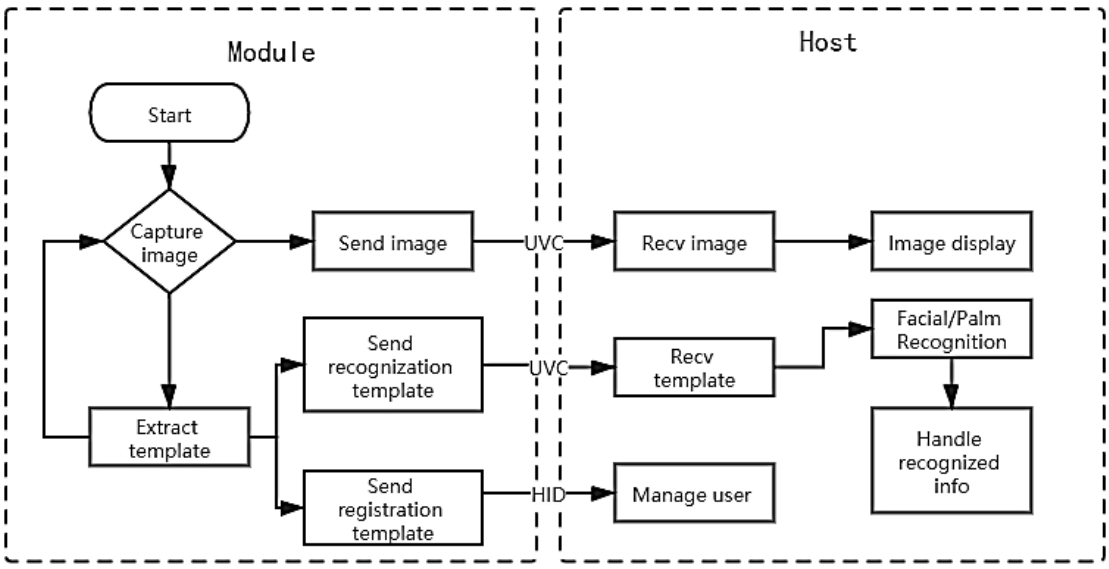
By providing a complete and rich SDK, developers can call and configure the corresponding functional interface according to their requirements. They can develop their applications and also completely integrate face or palm recognition functions on the client's host side, which considerably shortens the 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 is completed by it.
- The user collects the images through [AMTCameraLib] or other UVC implementation methods.
- 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, transmits the visible light image and the near-infrared image through their UVC ports, where the developers can select the required port settings according to the image requirements.

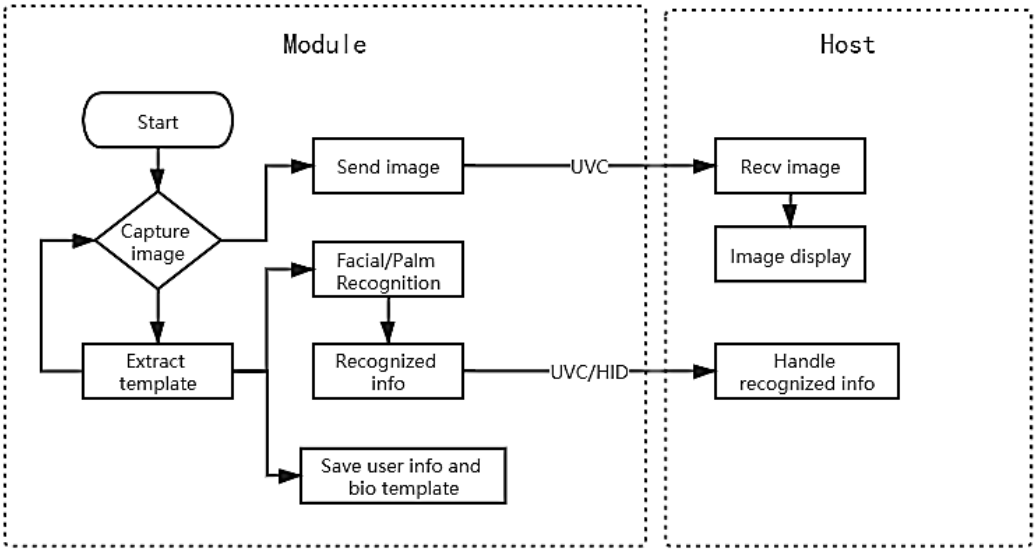
2.2.2 Match-on-Host and User Management

- If template matching and user management are performed on the host side, the module pushes 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 matching through [AMTFaceMatch] and [AMTPalmMatch].
- This method is suitable 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 template matching and user management within the module, then the module will generate the output of the matching result through the UVC protocol.
- After the matching 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 Match-on-Module and user management is as follows.



2.3 Rapid Integration

2.3.1 SDK File

DLL Contents

File Name	Description
AMTCameraLib.dll	UVC capture API dynamic link library
AMTHIDLib.dll	HID communication API dynamic link library
AMTFaceMatch.dll	Face recognition API dynamic link library
AMTPalmMatch.dll	Palm recognition API dynamic link library

AMTPalmVeinServer.dll	Palm recognition low-level API dynamic link library

2.3.2 Development Setup

SDK dynamic link library files can be copied and installed directly.

Before installing the AMTMultiBio SDK, please make sure that your operating system, computer configuration, or Windows mobile terminal device meets the operational requirements of the software.

Copy AMTCameraLib.dll, AMTHIDLib.dll, AMTFaceMatch.dll, AMTPalmMatch.dll, AMTPalmVeinserver.dll, and other DLL files from AMTMultiBio SDK to the user-specified path.

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

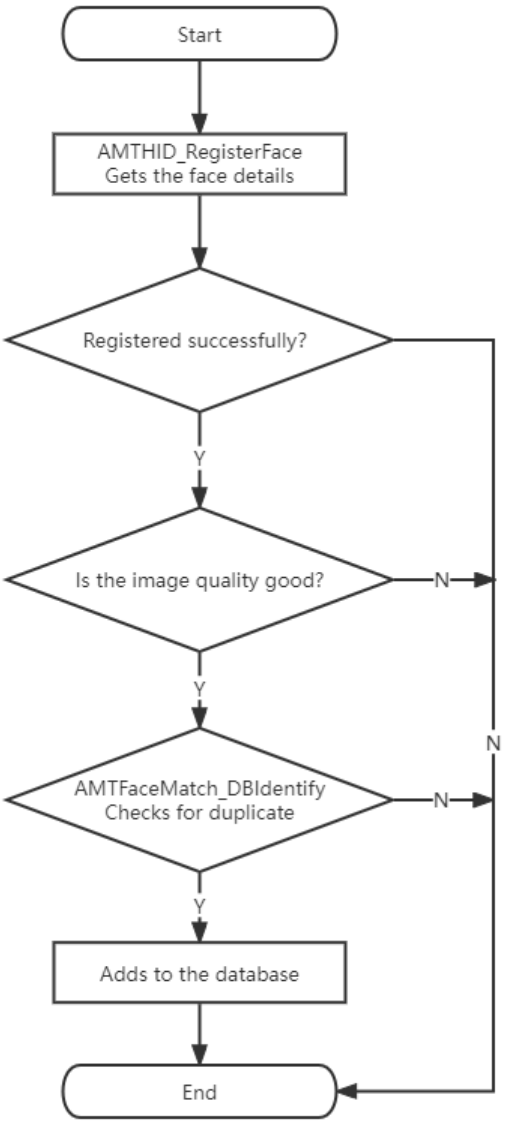
2.4 Programming Guide

Users can quickly understand the registration and matching process of faces and palms on the Host side through [Match-on-Host and User Management].

And by referring to [Match-on-Module Process], users can easily understand the process of internal staff management and Match-on-Module of the module.

2.4.1 Match-on-Host and User Management

Face Registration Process Flow

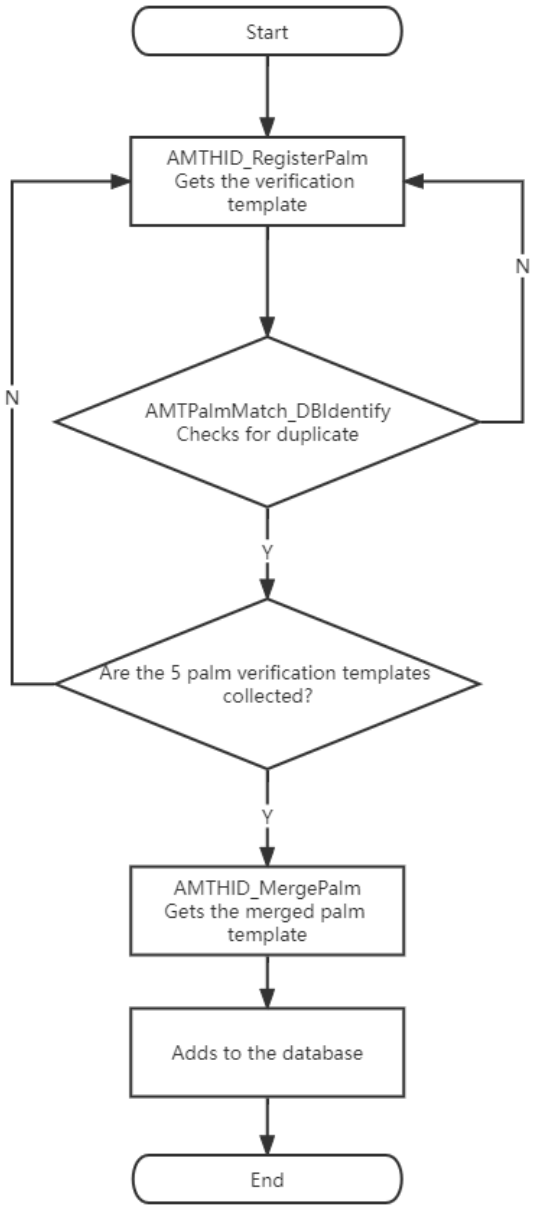


Process Description

- AMTHID_RegisterFace is passed in an image containing a visible face or captures a face image directly from within the module.
- The returned face information contains the angle of the face (recommended yaw<=25, pitch<=25, roll<=25), size (recommended faceWidth>=128, faceHeight>=128), and quality (recommended >=0.8). This information determines if the registered face is efficient and qualified.
- The AMTFaceMatch_DBIdentify function checks for the matching template.

- And the matching template is added to the algorithm cache by AMTFaceMatch_DBAdd.

Palm Registration Process Flow



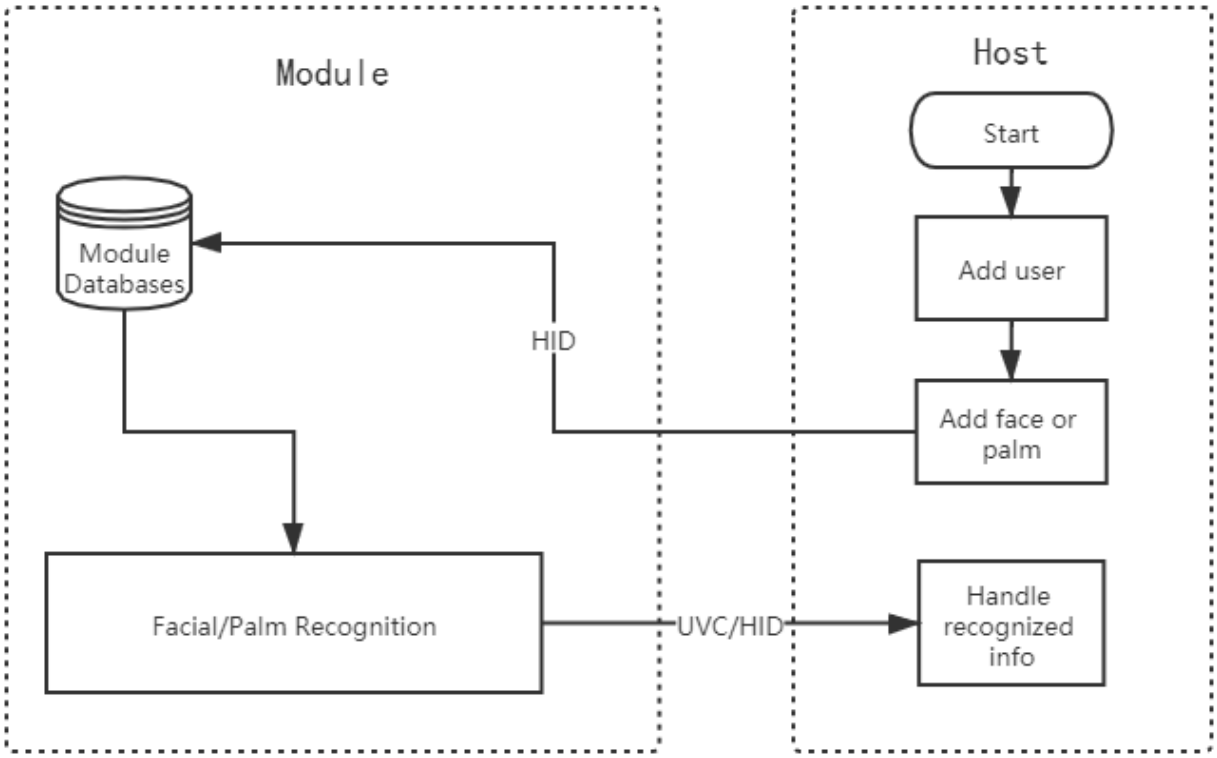
Process Description

- AMTHID_RegisterPalm is passed in an 8-bit grayscale image or uses the palm image captured directly from within the module.
- The AMTPalmMatch_DBIdentify function gets the verification template and checks for duplication.
- Then, the AMTHID_MergePalm interface is called to merge the collected five templates as a registration template.

- The merged registration template is then added to the algorithm cache by AMTPalmMatch_DBAdd.

2.4.2 Match-on-Module Process

- When the user directly registers the face/palm in the module, the personnel information and the face/palm template will be stored in the module.
- And the required personnel information will acquire through UVC or [Polling Recognition Result].
- The flowchart of the module's internal registration is as follows.



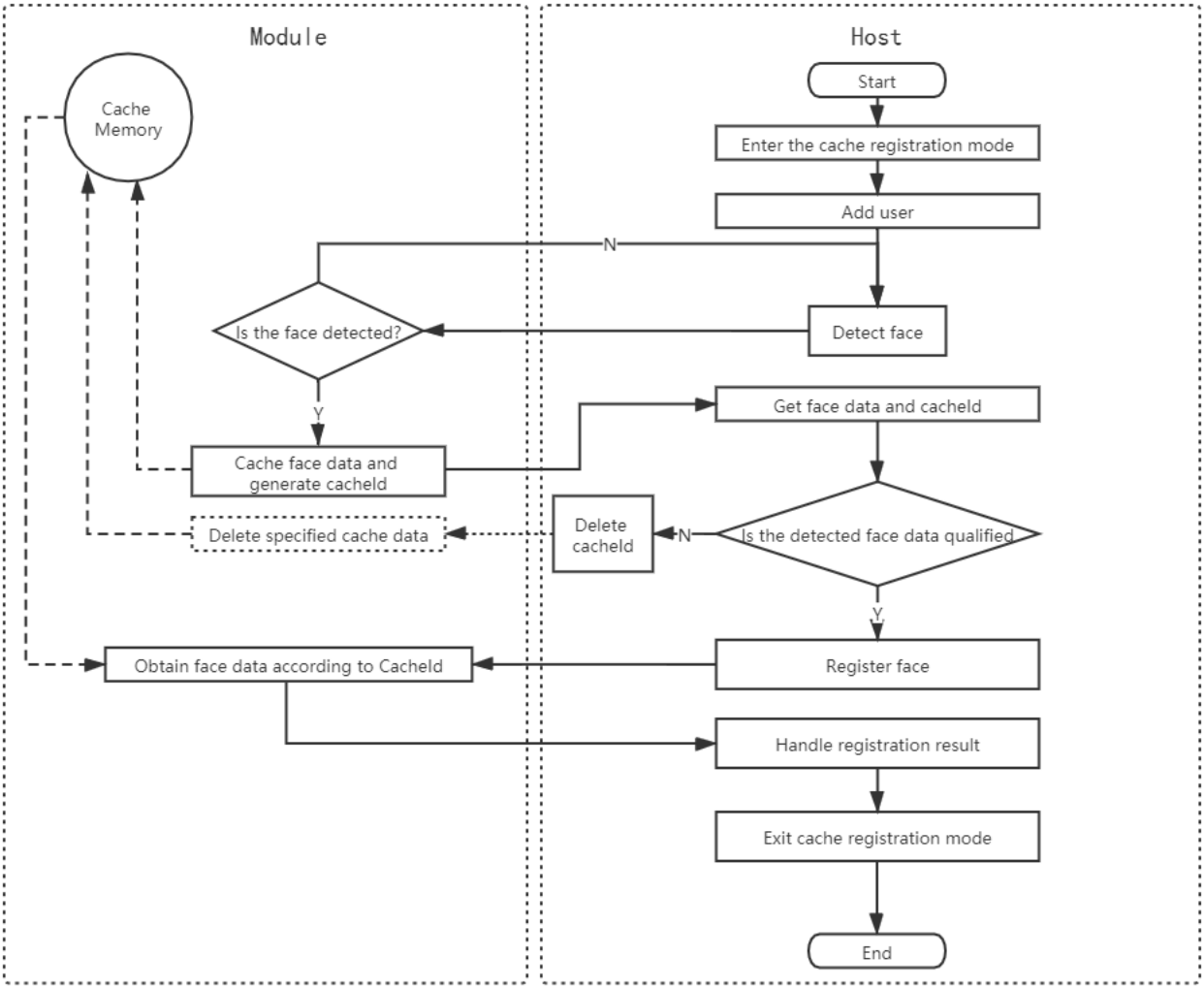
Note: If the developer uses the polling recognition result, then it is necessary to suspend HID polling when using the HID command to obtain other data or else it will affect the acquisition of the data.

Process Description

- To use the internal matching 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.

2.4.2.1 Face Cache Registration

When using the internal matching mode, the registered face will get authenticated via the internal CacheId in cache registration mode. Refer to the following process.

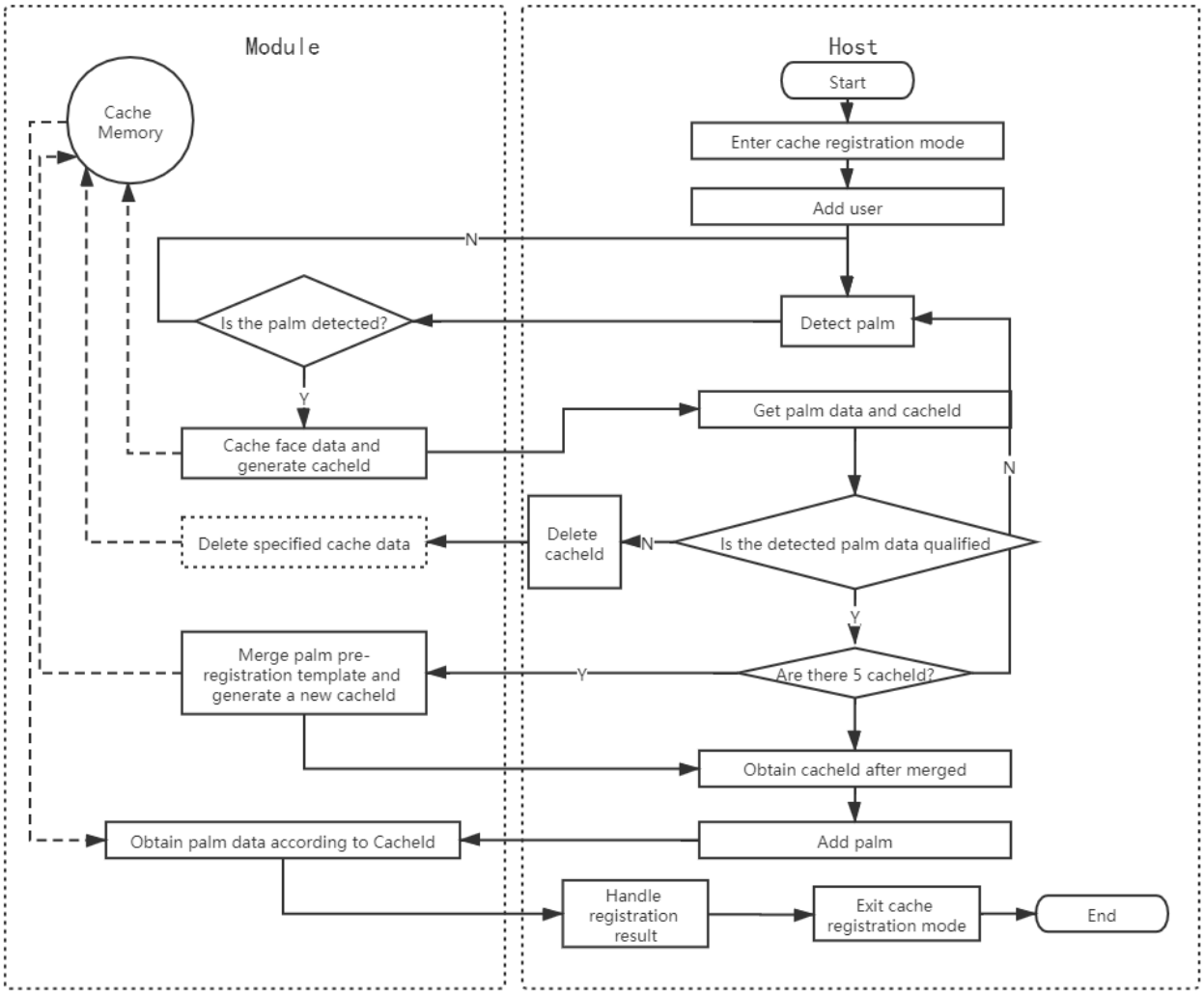


Process Description

- 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 completed, it is necessary to exit the [Cache Registration] mode or else the face and palm cannot be compared.

2.4.2.2 Palm Cache Registration

When using the internal matching mode, the registered palm will get authenticated via 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 Capture API

AMTCameraLib.dll is a dynamic link library of UVC acquisition API, primarily used to obtain UVC video streams and custom data.

Function List

Interface	Description
VideoData	Video data structure
CustomData	Custom data structure
VideoDataCallback	Pointer to video callback function
CustomDataCallback	Pointer to custom data callback function
AMTCamera_Init	Initializes the device
AMTCamera_Terminate	Terminates the device
AMTCamera_GetDeviceCount	Gets the total count of Devices
AMTCamera_GetDeviceType	Gets the details of the Device type
AMTCamera_OpenDevice	Opens the device
AMTCamera_CloseDevice	Closes the device
AMTCamera_GetCapWidth	Gets the image width
AMTCamera_GetCapHeight	Gets the image height
AMTCamera_SetDataCallback	Sets up the video and custom data callback
AMTCamera_FreePointer	Frees the memory pointer allocated during the callback

VideoData

Function Syntax

```
{
    uint32_t frame_index;
    uint32_t ori_length;
    uint32_t data_length;
    unsigned char* data;
        };
```

Parameters

Parameter	Description
uint32_t frame_index	It is the video frame number.
uint32_t ori_length	It is the original data length (jpeg data integrity verification)
uint32_t data_length	It is the actual data length.
unsigned char* data	It is the data buffer, that needs to be released by the user by executing AMTCamera_FreePointer function after use.

Remarks

- Click here to view the Function List.

CustomData

Function Syntax

```
{
    int64_t frame_index;
    int32_t width;
    int32_t height;
    char *customData;
        };
```

Parameters

Parameter	Description
int64_t frame_index	It corresponds to the video frame number in VideoData.
int32_t width	Image width

int32_t height	Image height
char *customData	It is the data buffer, that needs to be released by user by executing AMTCamera_FreePointer function. For custom JSON data see communication data protocols (JSON Data Definition)

Remarks

- For custom JSON format data with information about face tracking, face attributes, liveness, face templates, palm detection, palm templates, and so on, refer to Face Tracking and Recognition and Palm Recognition.
- Click here to view the Function List.

VideoDataCallback

Function Syntax

```
typedef void(__stdcall *VideoDataCallback)
                    (
                        void *pUserParam,
                        VideoData data
                            )
```

Description

Pointer to video callback function.

Parameters

Parameter	Description
pUserParam	**Out**: User custom data passed in when calling AMTCamera_SetDataCallback function.
data	**Out**: Video data structure

Remarks

- Click here to view the Function List.

CustomDataCallback

Function Syntax

typedef void(__stdcall *CustomDataCallback)
(
void *pUserParam,
CustomData data
)

Description

Pointer to custom data callback function.

Parameters

Parameter	Description
pUserParam	**Out**: User custom data passed in when calling AMTCamera_SetDataCallback function
data	**Out**: Custom data structure

Remarks

- Click here to view the Function List.

AMTCamera_Init

Function Syntax

int __stdcall AMTCamera_Init()

Description

Initializes the devices.

Returns

0	Success
Other	Failure

Remarks

- Click here to view the Function List.

AMTCamera_Terminate

Function Syntax

int __stdcall AMTCamera_Terminate()

Description

Terminates the devices.

Returns

0	Success
Other	Failure

Remarks

- Click here to view the Function List.

AMTCamera_GetDeviceCount

Function Syntax

int __stdcall AMTCamera_GetDeviceCount()

Description

Gets the total count of the devices.

Returns

>0	Number of devices
Other	Failure

Remarks

- Click here to view the Function List.

AMTCamera_GetDeviceType

Function Syntax

int __stdcall AMTCamera_GetDeviceType(int index)

Description

Gets the details of the device type.

Parameters

Parameter	Description
index	**In**: Device index 0~(n-1), n=AMTCamera_GetDeviceCount

Returns

1	Visible light camera
2	NIR camera
<0	Failed

Remarks

- Click here to view the Function List.

AMTCamera_OpenDevice

Function Syntax

```
int __stdcall AMTCamera_OpenDevice
        (
                int index,
                int w,
                int h,
                int fps,
                void **handle
                        )
```

Description

Opens the device.

Parameters

Parameter	Description
index	**In**: Device index 0~(n-1), n=AMTCamera_GetDeviceCount
w	**In:** Image width
h	**In**: Image height
fps	**In**: Frame rate, 25 or 30 frames
handle	**Out:** Handle of the device

Returns

0	Success
Other	Failure

Remarks

- Supported resolutions: 480*640, 720*1280.
- Supported frame rates: 25, 30.
- Click here to view the Function List.

AMTCamera_CloseDevice

Function Syntax

int __stdcall AMTCamera_CloseDevice(void *handle)

Description

Closes the device.

Parameters

Parameter	Description
handle	**In:** Handle of the device

Returns

0	Success
Other	Failure

Remarks

- Click here to view the Function List.

AMTCamera_GetCapWidth

Function Syntax

int __stdcall AMTCamera_GetCapWidth(void *handle)

Description

Gets the image width.

Parameters

Parameter	Description
handle	**In:** Handle of the device

Returns

Returns the image width.

Remarks

- Click here to view the Function List.

AMTCamera_GetCapHeight

Function Syntax

int __stdcall AMTCamera_GetCapHeight(void *handle)

Description

Gets the image height.

Parameters

Parameter	Description
handle	**In:** Handle of the device

Returns

Returns the image height.

Remarks

- Click here to view the Function List.

AMTCamera_SetDataCallback

Function Syntax

```
void __stdcall AMTCamera_SetDataCallback
        (
                void *handle,
                VideoDataCallback videoCallback,
                CustomDataCallback customCallback,
                void *pUserParam
                        )
```

Description

Sets up the video and the custom data callbacks.

Parameters

Parameter	Description
handle	**In:** Handle of the device
videoCallback	**In**: Pointer to video callback function
customCallback	**In:** Pointer to custom data callback function
pUserParam	**In:** User custom data, returned via callback function

Remarks

- Click here to view the Function List.

AMTCamera_FreePointer

Function Syntax

void __stdcall AMTCamera_FreePointer(void *pPointer)

Description

Frees the memory pointer allocated during the callback.

Parameters

Parameter	Description
pPointer	**In:** pointer passed from callback function

Remarks

- Click here to view the Function List.

3.2 HID Communication Interface

AMTHIDLib.dll is a dynamic link library of HID communication API, primarily used to read, set parameters, upgrade, manage users, etc.

Function List

Interface	Description
SendFileCallback	Pointer to SendFileCallback function
AMTHID_Init	Initializes the HID device
AMTHID_Terminate	Terminates the HID device
AMTHID_GetCount	Gets the total count of HID Devices
AMTHID_Open	Opens the HID device
AMTHID_Close	Closes the HID device
AMTHID_GetConfig	Gets the configuration Information
AMTHID_SetConfig	Sets the configuration information
AMTHID_RegisterFace	Registers the Face

AMTHID_RegisterPalm	Registers the Palm and extracts the templates.
AMTHID_MergePalm	Merges the extracted template of the registered Palm.
AMTHID_SnapShot	Captures an image
AMTHID_SendFile	Sends the file
AMTHID_Reboot	Reboots the device
AMTHID_ManageModuleData	Manages the module internal data, including user management, biometric templates, matching records, etc.
AMTHID_PollMatchResult	Polls recognition result

SendFileCallback

Function Syntax

```
typedef void(__stdcall *SendFileCallback)
                    (
                        void *pUserParam,
                        int progress
                        )
```

Description

Pointer to SendFileCallback function.

Parameters

Parameter	Description
pUserParam	**Out:** User custom data passed in when calling AMTHID_SendFile function
progress	**Out:** Sends the file progress, and its range is 0-100

Remarks

- Click here to view the Function List.

AMTHID_Init

Function Syntax

int __stdcall AMTHID_Init()

Description

Initializes the HID device.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- Click here to view the Function List.

AMTHID_Terminate

Function Syntax

int __stdcall AMTHID_Terminate()

Description

Terminates the HID device.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- Click here to view the Function List.

AMTHID_GetCount

Function Syntax

int __stdcall AMTHID_GetCount(int *count)

Description

Gets the total count of HID Devices.

Parameters

Parameter	Description
count	**Out**: Returns the total count of HID devices.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- Click here to view the Function List.

AMTHID_Open

Function Syntax

int __stdcall AMTHID_Open(int index, void **handle)

Description

Opens the HID device.

Parameters

Parameter	Description
index	**In**: Device index 0~(n-1), n=AMTHID_GetCount
handle	**Out:** Handle of the device

Returns

0	Success

Other	Failure (see Appendix 1)

Remarks

- Click here to view the Function List.

AMTHID_Close

Function Syntax

int __stdcall AMTHID_Close(void *handle)

Description

Closes the HID device.

Parameters

Parameter	Description
handle	**In:** Handle of the device

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- Click here to view the Function List.

AMTHID_GetConfig

Function Syntax

```
int __stdcall AMTHID_GetConfig
        (
                void *handle,
                int type,
                char *json,
```

```
    int *len
        )
```

Description

Gets the configuration parameters.

Parameters

Parameter	Description
handle	**In:** Handle of the device
type	**In:** 1 COMMON_ CONFIG 2 CAPTURE_FILTER_ CONFIG 3 MOTION_DETECT_CONFIG 4 PALM_ CONFIG 5 DEVICE_INFORMATION 6 DEVICE_TIME
json	**Out**: Returns the JSON data
len	**In:** Memory size allocated to JSON; it is recommended to allocate 2K bytes.
	Out: Length of the actual output of the JSON data

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to [Get and set configuration parameters].
- Click here to view the Function List.

AMTHID_SetConfig

Function Syntax

```
int __stdcall AMTHID_SetConfig
        (
            void *handle,
```

```
int type,
char *json
)
```

Description

Sets the configuration parameters.

Parameters

Parameter	Description
handle	**In:** Handle of device
type	**In:** 1 COMMON_ CONFIG 2 CAPTURE_FILTER_ CONFIG 3 MOTION_DETECT_CONFIG 4 PALM_ CONFIG 6 DEVICE_TIME
json	**In**: Sets the configured JSON data

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to [Get and set configuration parameters].
- Click here to view the Function List.

AMTHID_RegisterFace

Function Syntax

```
int __stdcall AMTHID_RegisterFace
(
    void *handle,
    const char* json,
    char *faceData,
    int *len
```

)

Description

Registers the Face

Parameters

Parameter	Description
handle	**In:** Handle of the device
json	**In:** JSON string
faceData	**Out:** Returns the face registration template of the JSON data. The recommended allocation is 20*1024 bytes.
len	**In:** Length of the faceData
	Out: Returns the actual length of face registration result.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to Host side face registration process and [Face service-related functions].
- Click here to view the Function List.

AMTHID_RegisterPalm

Function Syntax

```
int __stdcall AMTHID_RegisterPalm
        (
                void *handle,
                const char* json,
                char *palmData,
                int *len
                        )
```

Description

Registers the Palm and extracts the templates.

Parameters

Parameter	Description
handle	**In:** Handle of the device
json	**In:** JSON string
palmData	**Out**: Returns the palm pre-registration and verification/identification template in JSON data format, and the recommended allocation is 200*1024 bytes.
len	**In**: Length of the palmData
	Out: Returns the actual length of the palmData

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to Host side palm registration process and Palm service-related functions.
- Click here to view the Function List.

AMTHID_MergePalm

Function Syntax

```
int __stdcall AMTHID_MergePalm
        (
                void *handle,
                char *palmData,
                int *len
                        )
```

Description

Merges the extracted template of the registered Palm.

Parameters

Parameter	Description
handle	**In:** Handle of the device

palmData	**Out:** Returns the palm registration template in JSON data format, and the recommended allocation is 20*1024 bytes.
len	**In:** Size of the palmData
	Out: Returns the actual size of the palmData

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to Merge palm pre-registration template.
- Call AMTHID_RegisterPalm to successfully obtain the verification/identification template 5 times before obtaining the palm registration template.
- Click here to view the Function List.

AMTHID_SnapShot

Function Syntax

```
int __stdcall AMTHID_SnapShot
        (
                void *handle,
                int snapType,
                char* snapData,
                int *size
                        )
```

Description

Captures an image.

Parameters

Parameter	Description
handle	**In:** Handle of the device
snapType	**In**: Snapshot type, snapshot RGB camera or NIR camera image
snapData	**Out**: Returns the captured JSON data, and the recommended allocation is 2*1024*1024 bytes
size	**In:** Size of the allocated snapData
	Out: Returns the actual length of the data.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to [SnapShot].
- Click here to view the Function List.

AMTHID_SendFile

Function Syntax

```
int __stdcall AMTHID_SendFile
        (
                void *handle,
                int fileType,
                char *filePath,
                SendFileCallback sendFileCallback,
                void *pUserParam
                        )
```

Description

Sends the files, upgrades the firmware, images, etc

Parameters

Parameter	Description
handle	**In:** Handle of the device
fileType	**In**: 0 Upgrade Image 1 Upgrade firmware
filePath	**In**: File path
sendFileCallback	**In**: Callback function, returns the progress of sending files
pUserParam	**In**: User custom data, returned via callback function

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- When sending large files such as images, it is recommended that the application should be implemented in a thread.
- Click here to view the Function List.

AMTHID_Reboot

Function Syntax

```
int __stdcall AMTHID_Reboot
                    (
                        void *handle,
                        int mode
                        )
```

Description

Reboots the device.

Parameters

Parameter	Description
handle	**In:** Handle of device
mode	**In**: 0 – Normal reboot 1 – Reboot to upgrade image mode

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- When sending the large files such as images, it is recommended that the application should be implemented in a thread.

- Click here to view the Function List.

AMTHID_ManageModuleData

Function Syntax

```
int __stdcall AMTHID_ManageModuleData
                    (
                        void *handle,
                        int type,
                        char *json,
                        char *result,
                        int *len
                            )
```

Description

Manages module internal data, including user management, biometric templates, matching records, etc.

Parameters

Parameter	Description
handle	**In:** Handle of the device
type	**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
json	**In:** JSON data is passed in when managing the users inside the module
result	**Out:** Returns the JSON data
len	**In:** The memory size allocated by **result**
	Out: Actual output result data size

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to [Module data management_snapShot].
- Click here to view the Function List.

AMTHID_PollMatchResult

Function Syntax

```
int __stdcall AMTHID_PollMatchResult
                    (
                        void *handle,
                        char *json,
                        int *len
                            )
```

Description

Polls recognition result

Parameters

Parameter	Description

handle	**In:** Handle of device
json	**Out:** Returns recognition result
len	**In:** The memory size allocated for JSON data; and the recommended size is 40*1024 bytes.
	Out: Returns the actual output size of the JSON data.

Returns

0	Success
Other	Failure (see Appendix 1)

Remarks

- For JSON data, please refer to [Polling recognition result_snapShot].
- Click here to view the Function List.

3.3 AMTFaceMatch

AMTFaceMatch.dll is a dynamic link library of face algorithm API, which implements 1:1 and 1:N verification/identification functions.

Function List

Interface	Description
AMTFaceMatch_GetVersion	Gets the face algorithm version
AMTFaceMatch_Init	Initializes the face algorithm
AMTFaceMatch_Terminate	Releases the face algorithm
AMTFaceMatch_Verify	1:1 Face verification
AMTFaceMatch_DBAdd	Adds the face template to the database.
AMTFaceMatch_DBDel	Deletes the specified face template from the database
AMTFaceMatch_DBClear	Clears the database.
AMTFaceMatch_DBCount	Gets the total count of the templates stored in the database
AMTFaceMatch_DBVerify	Verifies the captured template with the template of that specified ID stored in the database and returns the verification value.

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

AMTFaceMatch_GetVersion

Function Syntax

```
int __stdcall AMTFaceMatch_GetVersion
        (
                char* version,
                int *size
                        )
```

Description

Gets the face algorithm version number.

Parameters

Parameter	Description
version	**Out:** Returns the Face algorithm version number. Recommended space allocation is 64 bytes.
size	**In**: The amount of version space allocated.
	Out: Returns the actual version size.

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_Init

Function Syntax

```
int __stdcall AMTFaceMatch_Init(void** context)
```

Description

Initializes the face algorithm.

Parameters

Parameter	Description
context	**Out:** Face algorithm handle

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_Terminate

Function Syntax

int __stdcall AMTFaceMatch_Terminate(void *context)

Description

Releases the face algorithm.

Parameters

Parameter	Description
context	**In:** Face algorithm handle

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_Verify

Function Syntax

```
int __stdcall AMTFaceMatch_Verify
        (
                void *context,
                unsigned char *regTemplate,
                unsigned char *verTemplate,
                float *score
                        )
```

Description

1:1 Face verification.

Parameters

Parameter	Description
context	**In:** Face algorithm handle
regTemplate	**In**: Registration template
verTemplate	**In**: Face template
score	**Out**: Returns the verification score

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.
- Recommended threshold is 0.899.

AMTFaceMatch_DBAdd

Function Syntax

```
int __stdcall AMTFaceMatch_DBAdd
        (
                void *context,
                char *id,
```

```
                    unsigned char *regTemplate
                        )
```

Description

Adds the face template to the database.

Parameters

Parameter	Description
context	**In:** Face algorithm handle
id	**In:** Face ID
regTemplate	**In**: Registration template

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_DBDel

Function Syntax

```
int __stdcall AMTFaceMatch_DBDel
                        (
                            void *context,
                            char *id
                                )
```

Description

Deletes the specified face template from the database.

Parameters

Parameter	Description
context	**In:** Face algorithm handle
id	**In:** Face ID

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_DBClear

Function Syntax

int __stdcall AMTFaceMatch_DBClear(void *context)

Description

Clears the cache template.

Parameters

Parameter	Description
context	**In:** Face algorithm handle

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_DBCount

Function Syntax

```
int __stdcall AMTFaceMatch_DBCount
                    (
                        void *context,
                        int *size
                            )
```

Description

Gets the total count of the face template stored in the database.

Parameters

Parameter	Description
context	**In:** Face algorithm handle
size	**Out:** Number of face templates in the database

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.

AMTFaceMatch_DBVerify

Function Syntax

```
int __stdcall AMTFaceMatch_DBVerify
        (
                void *context,
                unsigned char *verTemplate,
                char *id,
                float *score
                        )
```

Description

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

Parameters

Parameter	Description
context	**In:** Face algorithm handle
verTemplate	**In:** Verification template
id	**In**: Face ID
score	**Out:** Returns the verification score

Returns

Error code	see Appendix 2

Remarks

- Click here to view the Function List.
- Recommended threshold is 0.899.

AMTFaceMatch_DBIdentify

Function Syntax

```
int __stdcall AMTFaceMatch_DBIdentify
        (
                void *context,
                unsigned char *verTemplate,
                char **id,
                float *score
                        )
```

Description

Identifies the captured template against all the stored templates in the database.

Parameters

Parameter	Description
context	**In:** Face algorithm handle
verTemplate	**In:** Identification template
id	**Out**: Returns the face ID
score	**Out:** Returns the identification score

Returns

Error code	(see Appendix 2)

Remarks

- Click here to view the Function List.
- Recommended threshold is 0.899.

3.4 AMTPalmMatch

AMTPalmMatch.dll is a palm algorithm API dynamic link library, which implements 1:1 and 1:N

verification/identification functions.

Function List

Interface	Description
AMTPalmMatch_GetVersion	Get the palm algorithm version
AMTPalmMatch_Init	Initialize the palm algorithm
AMTPalmMatch_Terminate	Release the palm algorithm
AMTPalmMatch_Verify	Perform 1:1 palm verification process
AMTPalmMatch_DBAdd	Add the palm template to the database
AMTPalmMatch_DBDel	Delete the specified palm template from the database
AMTPalmMatch_DBClear	Clear the palm templates from the database
AMTPalmMatch_DBCount	Get the total count of the palm templates stored in the database
AMTPalmMatch_DBVerify	Verify the captured template with the template of that specified ID stored in the database and returns the verification value.
AMTPalmMatch_DBIdentify	Identify the captured template against all the stored templates in the database and returns the identification value.

AMTPalmMatch_GetVersion

Function Syntax

```
int __stdcall AMTPalmMatch_GetVersion
        (
                char* version,
                int size

                        )
```

Description

Gets the palm algorithm version number.

Parameters

Parameter	Description
version	**Out:** Palm algorithm version number
size	**In**: The space size requested by the version, the recommended size to be allocated is 64 bytes

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_Init

Function Syntax

int __stdcall AMTPalmMatch_Init(void **context)

Description

Initializes the palm algorithm.

Parameters

Parameter	Description
context	**Out:** Palm algorithm handle

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_Terminate

Function Syntax

int __stdcall AMTPalmMatch_Terminate(void *context)

Description

Releases the palm algorithm.

Parameters

Parameter	Description
context	**In:** Palm algorithm handle

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_Verify

Function Syntax

```
int __stdcall AMTPalmMatch_Verify
        (
                void *context,
                unsigned char *regTemplate,
                unsigned char *verTemplate,
                int *score
                        )
```

Description

1:1 Palm Verification

Parameters

Parameter	Description
context	**Out:** Palm algorithm handle
regTemplate	**In:** Registration template

verTemplate	**In:** Verification template
score	**Out:** Returns the verification score

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.
- Recommended threshold is 576.

AMTPalmMatch_DBAdd

Function Syntax

```
int __stdcall AMTPalmMatch_DBAdd
        (
                void *context,
                const char *id,
                const unsigned char *regTemplate
                        )
```

Description

Adds the palm template to the database

Parameters

Parameter	Description
context	**In:** Palm algorithm handle
id	**In**: Palm ID
regTemplate	**In**: Registration template

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_DBDel

Function Syntax

```
int __stdcall AMTPalmMatch_DBDel
        (
                void *context,
                char *id

                        )
```

Description

Deletes the specified palm template from the database

Parameters

Parameter	Description
context	**In:** Palm algorithm handle
id	**In**: Palm ID

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_DBClear

Function Syntax

```
int __stdcall AMTPalmMatch_DBClear(void *context)
```

Description

Clears the database.

Parameters

Parameter	Description
context	**In:** Palm algorithm handle

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_DBCount

Function Syntax

```
int __stdcall AMTPalmMatch_DBCount
        (
                void *context,
                int *size
                        )
```

Description

Gets the total count of palm template stored in the database.

Parameters

Parameter	Description
context	**In:** Palm algorithm handle
size	**Out**: Number of palm templates stored in the database

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.

AMTPalmMatch_DBVerify

Function Syntax

```
int __stdcall AMTPalmMatch_DBVerify
        (
                void *context,
                unsigned char *verTemplate,
                const char *id,
                int *score

                        )
```

Description

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

Parameters

Parameter	Description
context	**In:** Palm algorithm handle
verTemplate	**In**: Verification template
id	**In**: Palm ID
score	**Out**: Returns the verification score

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.
- Recommended threshold is 576.

AMTPalmMatch_DBIdentify

Function Syntax

```
int __stdcall AMTPalmMatch_DBIdentify
        (
                void *context,
                unsigned char *verTemplate,
                char *id,
                int *score

                        )
```

Description

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

Parameters

Parameter	Description
context	**In:** Palm algorithm handle
verTemplate	**In**: Identification template
id	**Out**: Returns palm ID
score	**Out**: Returns the identification score

Returns

Error code	(see Appendix 3)

Remarks

- Click here to view the Function List.
- Recommended threshold is 576.

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

CacheId is mainly used for [Cache Registration], 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 faces, palms, etc.

4.1.3 Feature

Biometric template data can be used for face and palm recognition and registration.

The JSON data for biometric template 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 JSON 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 for recognition 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 is used to determine whether it is a photo or video attack.

The JSON data for face liveness 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, and its data type is float.

The JSON data for landmark is 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 JSON data for face tracking 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.

The 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.

The 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 for 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 for 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/IGq9gLAJvv+WhL39IgQ9AjCjvEAKAj4E/Eu8AWLgPH7ixr3/IoS9fq7vPYGw",
                "preTemplateSize": 98448
        }
}
```

Parameter Description

Parameter	Is it Required	Description
verTemplate	No	Palm verification/identification template, used for deduplication 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-Related Functions

4.2.1 Face Detection

Use HID's **AMTHID_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
                }
            }]
        },
        "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.

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 [CustomDataCallback]. 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 and identification information

liveness	No	Face tracking information, including face coordinates, etc.
tracker	No	Face quality
label	Yes	Biometric attribute category, 1 is face, 5 is palm

4.3 Palm-Related Functions

4.3.1 Palm Detection

The HID's **AMTHID_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	This parameter detects the palm image in JSON data format. If this parameter is not applied, the module will directly take a frame of the image from the near-infrared stream for palm detection by default.
feature	No	This parameter infers whether to return the palm template. If this parameter is not applied, the default value will be false.
palmInfo	No	This parameter infers whether to return palm information. If this parameter is not applied, the

		default will be false.
picture	No	This parameter infers whether to return registered palm image. If this parameter is not applied, the default will be 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 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 value details.

4.3.2 Merge Palm Pre-registration Template

Please notice, only the merged registration template is possible for registration. Users can call **AMTHID_MergePalm** 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.

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 gets 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	Biometric category, where 5 denotes palm
trackInfo	No	Palm **trackInfo** field, contains palm coordinates and image quality.

4.4 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 **AMTHID_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",
               "enableStoreAttendPhoto": false,
                "isTrackingMatchMode": true,
```

```
                "recogInterval": 1000,
                "recogRespirator": false,
                "recogThreshold": 0.89999997615814209,
                "scoringInterval": 5
        }
}
```

Call AMTHID_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,
                         "enableStoreAttendPhoto": false,
                        "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	Time interval for storing matching records
attrInterval	No	Time interval for detecting face attributes (unit: ms)

attributeRecog	No	Attribute recognition enable switch
countAlgorithm	No	Defines whether the firmware printing time consuming the algorithm is running.
debugLevel	No	Algorithm internal module debug level
drawTrackRect	No	UVC image drawing and tracking face frame
enableStoreAttendLog	No	Enables storing of matching log
enableStoreStrangerAttLog	No	Enables storing of unregistered matching log
faceAEEnabled	No	Area exposure enable switch
hacknessThreshold	No	Anti-fake threshold
isTrackingMatchMode	No	Defines its tracking match mode
enableStoreAttendPhoto	No	Enables storing photo
recogInterval	No	Time interval for recognition (unit: ms)
recogRespirator	No	Mask detection enable switch
recogThreshold	No	1:N Face identification 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.

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 [AMTHID_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 [AMTHID_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, where 0 means success. Please refer to [Appendix 4] for others.

4.4.3 Motion detection configuration

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

By calling MOTION_DETECT_CONFIG of [AMTHID_SetConfig], 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 [AMTHID_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	It is the brightness threshold [10~1000] indicates the maximum average brightness of the pixel. When this value exceeds, the fill-light will not get turned on; otherwise, the fill light will get turned on;
idleTimeOutMS	Yes	This function represents that after how many milliseconds, the fill light will get turned off if there is no biometric detection during the idle mode
motionDetectFunOn	Yes	This function indicates whether the function, motion detection control light is turned on or not; Value: "true"/"false";
sensitivityThreshold	Yes	It indicates the sensitivity threshold [0 ~ 100]. The smaller the value, the more sensitive it is;
status	Yes	Data reply status value, where 0 means success. Please refer to [Appendix 4] for others.

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 **AMTHID_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 [AMTHID_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	1:N palm recognition 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.

4.4.5 Device Information

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

Call the DEVICE_INFORMATION of [AMTHID_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	Serial number of cpID
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, where 0 means success. Please refer to [Appendix 4] for others.

4.4.6 Device Time Configuration

Developers can synchronize the device time by calling DEVICE_TIME of **AMTHID_SetConfig**.

Set the JSON data format as follows:

Function Syntax

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

DEVICE_TIME can be obtained through [AMTHID_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.

4.5 Operation Related

4.5.1 Snapshot

Through the HID command, the frame of the current image can be captured and return 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.

4.6 Module Data Management

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

4.6.1 Add User

Add a user to the module through [AMTHID_ManageModuleData].

The JSON data format for adding users is as follows:

Function Syntax

```
{
        "personId": "12345",
        "groupId": "DA640D7CE7544B33A3A1C81CD95D3E66 ",
        "userId": "12345",
        "name": "name1",
        "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
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

Function Syntax


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

4.6.2 Delete User

This function deletes a specific user from the module through DEL_PERSON of [AMTHID_ManageModuleData].

The JSON data format for delete 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:

Function Syntax

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

4.6.3 Clear Users

This function clears all the users in the module through the CLEAR command of [AMTHID_ManageModuleData].

The JSON data format for cleared user response is as follows:

Function Syntax

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

4.6.4 Query User

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

The JSON data format for 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
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, where 0 means success. Please refer to [Appendix 4] for others.

4.6.5 Query All Users

Through QUERY_ALL_PERSON of [AMTHID_ManageModuleData], the developer 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",
                        "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, where 0 means success. Please refer to [Appendix 4] for others.

4.6.6 Get Person Statistics

Through the QUERY_STATISTICS of [AMTHID_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, where 0 means success. Please refer to [Appendix 4] for others.

4.6.7 Matching Record Count

Through ATT_RECORD_COUNT of [AMTHID_ManageModuleData], the developer can query the count of the matching records in the module.

The JSON data format for querying matching record count is as follows.

Function Syntax

```
    {
        "startTime" : "2019-10-01 10:00:00",
```

```
        "endTime" : "2019-10-01 12:00:00"
        }
or
    {
        "startId" : -1
        }
```

Parameter Description

Parameter	Is it Required	Description
startTime	No	Start time
endTime	No	End time
startId	No	-1 means querying all matching record count

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

Function Syntax

```
{
    "detail" : "success",
    "startId" : 1,
    "status" : 0,
    "totalCount" : 1
}
```

Parameter Description

Parameter	Is it Required	Description
startId	Yes	Starting index of matching records
totalCount	Yes	Total count of matching records
status	Yes	Data reply status value, where 0 means success. Please refer to [Appendix 4] for others.

4.6.8 Export Matching Log Record

Through EXPORT_ATT_RECORD of [AMTHID_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 records
reqCount	Yes	Request count of matching records
needImg	Yes	Whether to export photos. The module does not store photo as default. Please enable enableStoreAttendPhoto in commonSettings once needed.

The JSON data format returned by exporting matching records 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.

4.6.9 Clear Matching log Record

Through CLEAR_ATT_RECORD of [AMTHID_ManageModuleData], the developer can clear all the matching records in the module.

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

Function Syntax

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

4.6.10 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.

The polling process is through [AMTHID_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	Biometric attribute category, 1 is face, 5 is palm
identify	Yes	Identification information

4.6.11 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:

Function Syntax

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

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 Response:

Function Syntax


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

4.6.12 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 Palm through Photos

Developers can add palm images 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
image	No	8-bit grayscale image for registering the palm

Add palm Response:

Function Syntax

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

2. Add Palm 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 Response:

Response Syntax

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

4.6.13 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:

```
{
        "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:

```
{
        "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.

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"
}
```

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 the data, otherwise the information will not be returned.
picture	No	The returned registered palm image. If the registration request is true, it will return the data, otherwise the information will not be returned.
palmInfo	No	The returned palm information. If the registration request is true, it will return the data, 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.

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:

```
{
        "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

The response data is:

```
{
    "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	One or more palm information returned
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.

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:

```
{
    "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
data	No	Associated with **cacheIds**

Add palm response:

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

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:

```
{
	"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

5 Appendix

Appendix 1: AMTHIDLib Error Code

Error Code	Description
0	Success
-1	Failed to open 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 sufficient.
-6	Failed to send command
-7	Failed to read data, timeout
-8	Setting failed
-9	Failed to open file
-10	Abnormal file
-11	Illegal protocol header
-12	Sum check failed
-13	File reading failed

Appendix 2: AMTFaceMatch Error Code

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

Appendix 3: AMTPalmMatch Error Code

Error Code	Description
0	Success
-1	Parameter error
-2	Failed to apply for memory allocation
-3	Not enough memory
-4	Unsupported Interface
-5	Failed to load algorithm library
-6	Failed to initialize the algorithm library
-103	Illegal 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 the file directory	mkdir directory failed
507	The server failed to read the file	Failed to read the file
508	Failed to write the file on the 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 the database file	Generally, the database file is invalid or incomplete.
514	Failed to load the feature vector	Generally, the feature of the vector file is invalid or incomplete.
515	Failed to calculate the MD5 value	Failed to calculate MD5 value of the file.
520	Failed to find the configuration file	Generally, when obtaining or setting the configuration file, the corresponding configuration

		parameter is not found
521	Failed to the 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 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 the 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