ARMATURA

API Development Manual:

AMTFaceLite SDK For Windows

API Version: 12.0

Doc Version: 1.0

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.

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.com.

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 **AMTFaceLite 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 represents a note that needs to pay more attention to.
💡	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 with basic SDK development guide and technical background to help with better use of our AMTFaceLite SDK document. 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 explain all the required information on how to perform and integrate AMTFaceLite SDK.

1.1 Overview of the SDK

AMTFaceLite SDK is a wrapper of Armatura near-Infrared light face recognition algorithm. It is an excellent near-infrared face recognition algorithm based on the indoor face recognition algorithm, developed to resist complex ambient light and the needs of large capacity recognition. In the case of ensuring a very low FRR, the algorithm focuses on improving the wide adaptation to the environment and user habits, thereby greatly improving the robustness and success rate of face recognition.

The SDK provides the rich interfaces to access the algorithm’s functionalities for face recognition process, including face detection, feature extraction, liveness detection, template creation, and face identification.

The FaceLite SDK utilizes the widely supported libusb API for face module communication, supports common-used operation systems, and frees the developers from intimidating hardware operations. It is a developer-friendly toolkit to empower the biometric features on the software application with easy pickup.

The simple library components aid in supporting and enhancing the security requirements through biometric facial recognition which avoids spoofing and has been widely used in various systems, including attendance, security, video monitoring, and so on.

1.2 Feature of the SDK

- **Face Focusing Method to Enhance Image Quality:**
  The FaceLite algorithm takes face focusing method to enhance the image quality which significantly reduces the facing-light and back-light impact on the captured image.

- **Stable Face Features Boost Recognition Accuracy and Performance**
  The FaceLite algorithm can detect different levels (18,40 or 120) of key face feature points and their positions in milliseconds, such as eyes, lips, nose tips, and contours. Such key points are stable face features and can be recognized from the deliberated and unintentional variations in the captured face images. It boosts the algorithm to achieve face recognition accuracy and performance.

- **Multi-dimensional Face Feature Template for Robust Face Recognition:**
  The FaceLite algorithm calculates multi-dimensional features from the collected multiple templates (5 consecutive templates) to generate one enrollment template which minimizes the side impact from hats, scarves, dark glasses, or other attachments during the registration process. This improves the recognition robustness.

- **Liveness Detection:**
  The FaceLite algorithm can effectively detect a fake face from a digital photo, printed color photo, Black & White face image, or a recorded video of a live face.

- **High Recognition Performance**
  Based on the stable face features, the FaceLite algorithm takes the multi-level matching mode with optimized classifier parameters to match the candidate in the large-volume template library within a second.

- **Automatic Update the Template Library**:
  The FaceLite algorithm tracks face features and automatically updates the face template into the template library, such an adaptive approach can keep the template stay with the user’s current appearance and lower the rejection rate caused by changes in the user's appearance and hairstyle.

- **Algorithm Integrity**:
  Combined with Armatura near-infrared light face module, the FaceLite algorithm ensures the quality of images by maintaining data integrity for a genuine and accurate image process.

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

Development Language

This SDK provides a standard Win32 API interface and supports C, C++, and C# language development.

Platform Requirements

This SDK supports 32-bit/64-bit operating systems with Windows XP SP3 or higher.

Technical Parameters

Parameter	Description
Template size	< 29 KB
Gesture adaptability	Yaw ≤25°, Pitch ≤25°, Roll ≤25°
Face detection	< 80 ms
Face feature extraction	< 100 ms
Face verification/identification (1:6000)	< 100 ms
Number of face templates supported	6000
Accuracy	FRR = 98.6% when FAR = 0.001%

The preceding algorithm capability indicators are all measured based on an actual image data set (resolution of 480 x 640), 8GB memory, and quad-core Inter(R) Core(TM) i5-3210M CPU @2.5GHz processor.

2.1 Architecture

2.1.1 SDK Files

Copy the following files (DLL directory) to the Windows terminal.

File Name	Description
face.dat	Algorithm model file
THFaceImage.dll	Dynamic link library for the algorithm interface
THFaceLive.dll	Dynamic link library for the algorithm interface
THFacialPos.dll	Dynamic link library for the algorithm interface

AMTInfraredFace.dll	Low-level algorithm interface dynamic library
AMTNIRFace.dll	Dynamic library of near-infrared face interface
AMTFaceCap.dll	Dynamic link library for underlying interfaces of face capturing process.
libamtsensorcore.dll	Dynamic link library for underlying communication interfaces of the device
sqlite3.dll	Dynamic link library containing the command-line tools used for managing the SQLite database

2.1.2 Development Setup

SDK dynamic library files can be copied and installed directly

Before installing AMTFaceLite SDK, please make sure that the operating system, system configuration, or Windows portable terminal device meets the requirements for software operation.

Copy related files such as AMTNIRFace.dll, AMTInfraredFace.dll, AMTFaceCap.dll, THFacialPos.dll, THFaceLive.dll, THFaceImage.dll, face.dat, libamtsensorcore.dll, sqlite3.dll and other related files from the AMTFaceLite SDK to the path specified by the user.

2.1.3 USB Information

USB dongle

The AMTNIRFACE12.0 algorithm uses a dongle for user authorization. The dongle is usually built into face recognition devices. Therefore, you do not need an external dongle.

2.2 Programming Guide

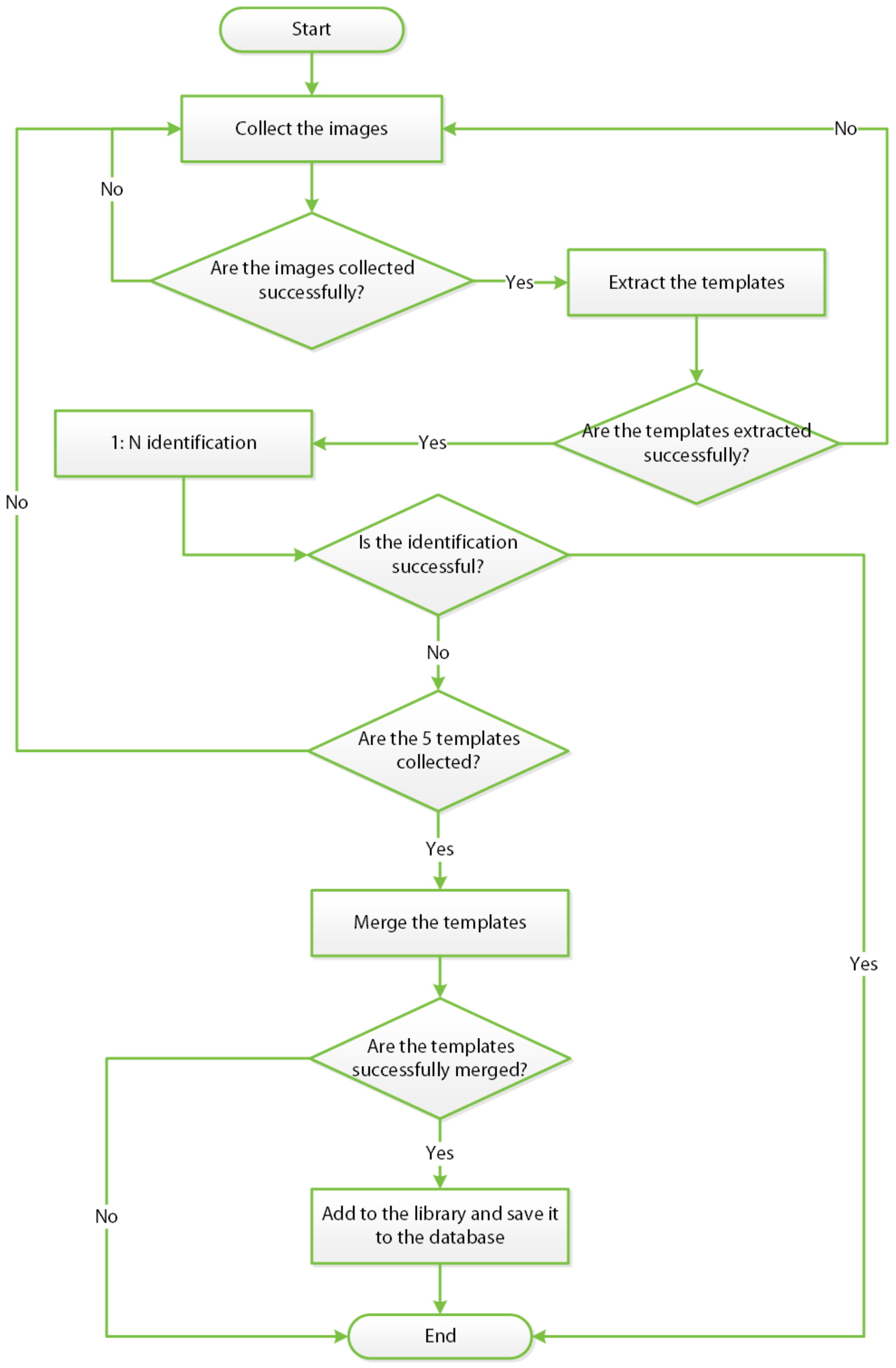
This section describes the key processes of face recognition to help developers understand the face registration and verification/identification processes implemented by the AMTNIRFACE12.0 algorithm.

2.2.1 Registration Process

In the face registration process, the face recognition application must capture five verification/identification templates and merge them into a registered template.

For more details about different types of templates, see the SDK Interface Description.

Registration Process Flow Using 1:N Identification


Start
Collect the images
No
Are the images collected successfully?
Yes
Extract the templates
No
Are the templates extracted successfully?
Yes
1: N identification
No
Is the identification successful?
No
Are the 5 templates collected?
Yes
Merge the templates
Yes
Are the templates successfully merged?
Yes
Add to the library and save it to the database
No
End

Process Description:

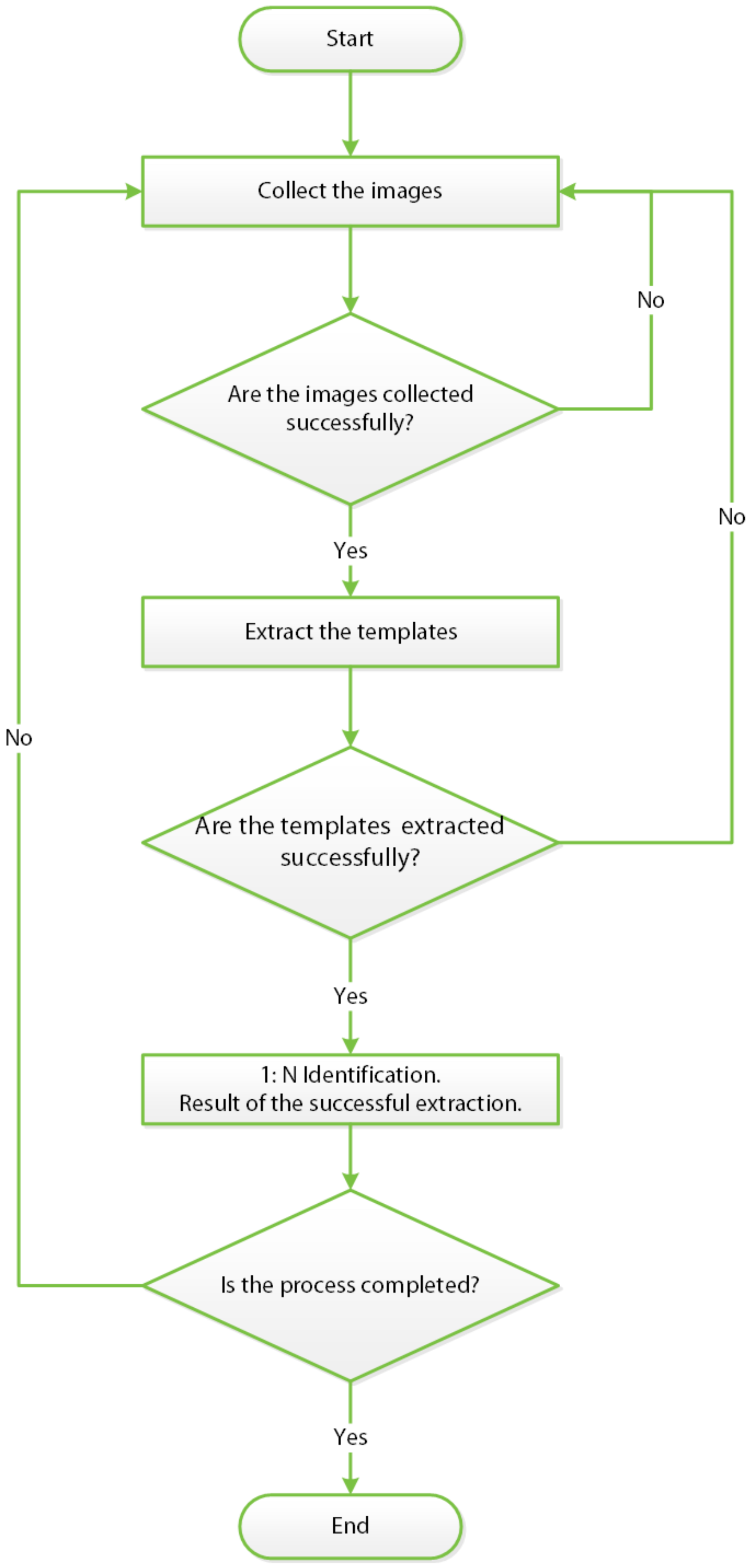
- The application calls the face capturing SDK to capture the face images.
- Once the face images are captured successfully, the application calls the extract function AMTNIRFace_ExtractFromGrayscaleData to extract the templates.
- Then the application calls the AMTNIRFace_DBIdentify 1:N function to determine whether the current extracted template has been registered.
- And, if it has been registered, the application returns a message and ends the registration process.
- And if less than five templates have been captured, the application continues to capture the next template.
- After capturing five templates, the application merges the templates into a registered template. If the registration fails, the application returns a message and ends the registration process.
- If the registration succeeds, the application calls the dbAdd function to add the registered template to the database.
- And thus, ends the process.

2.2.2 Verification/Identification Process

1:N Identification Process

To implement 1:N face identification, it is required to add all the registered templates to the database. It is recommended to call the AMTNIRFace_DBAdd function to add all registered templates to the database after successful algorithm initialization.

Identification Process Flow


Start
Collect the images
Are the images collected successfully?
No
Yes
Extract the templates
Are the templates extracted successfully?
No
Yes
1: N Identification.
Result of the successful extraction.
Is the process completed?
No
Yes
End

Process Description

- The application calls the face capturing SDK to capture the face images.
- After the face image is captured successfully, the application calls the AMTNIRFace_ExtractFromGrayscaleData function to extract a template.
- The application calls the AMTNIRFace_DBIdentify 1:N function to compare the current template with registered templates.
- And once the registered template is identified, the application ends the registration process.

3 SDK Interface Description

3.1 NIR Face Template Format Description

Template type	Data length	Description
Verification/Identification template	21072 bytes	Pre-registered or verification/identification template
Registered template	28992 bytes	Registered or Registration template

3.2 Near-Infrared Face Interface

AMTNIRFace.dll dynamic library is a dynamic library of Near Infrared Face Interface, mainly used for extraction, registration, and verification/identification of near-infrared face templates.

3.2.1 AMTNIRFace.dll

Function List

Function Interface	Description
AMTNIRFace_Version	Gets the SDK version number
AMTNIRFace_Init	Initializes the algorithm resources
AMTNIRFace_Terminate	Releases the algorithm resources
AMTNIRFace_ExtractFromGrayscaleData	Extracts a verification/identification template from 256-gray scale pixel data
AMTNIRFace_GetTemplateQlt	Gets the quality of a face verification/identification template
AMTNIRFace_Verify	Performs the 1:1 face verification
AMTNIRFace_DBVerifyByID	Performs 1:1 verification with the specified faceID
AMTNIRFace_MergeRegTemplate	Merges the verification/identification templates into a registered templates
AMTNIRFace_DBAdd	Adds the registered template to the database
AMTNIRFace_DBDel	Removes the specified face template from the database
AMTNIRFace_DBClear	Clears the database
AMTNIRFace_DBCount	Gets the total number of face templates stored in

	the database
AMTNIRFace_DBIdentify	Performs 1:N face Identification
AMTNIRFace_GetFacePosition	Gets the position coordinates of the near-infrared face
AMTNIRFace_DetectAndGetPos	Face detection and face position acquisition
AMTNIRFace_GetLiveness	Face live detection

AMTNIRFace_Version

Function Syntax

int __stdcall AMTNIRFace_Version(char* version, int* size);

Description

Gets the SDK version number.

Parameters

Parameter	Description
version	**Out**: Returns the version number (recommended to pre-allocate 128 bytes, enough to use)
size	**In:** Version memory size (bytes)
	Out: Returns the actual version length

Returns

See the Error Code

Example

```
char szVer[128] = {0};
int len = 128;
ret = AMTNIRFace_Version(szVer,&len);
......
```

Remarks

Click here to view the Function List.

AMTNIRFace_Init

Function Syntax

int __stdcall AMTNIRFace_Init(void** context);

Description

Initializes the algorithm resources.

Parameters

Parameter	Description
context	**Out**: Algorithm instance pointer

Returns

See the Error Code

Example

```
......
void* pInstanceContext = NULL;
ret = AMTNIRFace_Init(&pInstanceContext);
......
```

Remarks

Click here to view the Function List.

AMTNIRFace_Terminate

Function Syntax

int __stdcall AMTNIRFace_Terminate(void* context);

Description

Releases the algorithm resources.

Parameters

Parameter	Description
context	**In:** Algorithm instance pointer

Returns

See the Error Code

Remarks

Call this function at the end of the program.

Click here to view the Function List.

AMTNIRFace_ExtractFromGrayscaleData

Function Syntax

```
int __stdcall AMTNIRFace_ExtractFromGrayscaleData
        (
                void* context,
                unsigned char* rawImage,
                int width,
                int height,
                unsigned char* verTemplate,
                int *cbVerTemplate,
                int  expmode,
                int *exp
                        );
```

Description

Extracts a verification/identification template from 256-gray scale pixel data.

Parameters

Parameter	Description
context	**In:** Algorithm instance pointer
rawImage	**In:** Grayscale image bit depth 8-bit original image data (256-gray

	scale pixel data)
width	**In:** Image width
height	**In:** Image height
verTemplate	**Out:** Returns the face verification/identification template data
cbVerTemplate	**In:** vertmp memory allocation size
	Out: Returns the actual data length of verTemplate verification/identification template
expmode	**In:** Exposure mode (0 for registration, 1 for recognition)
exp	**Out:** Exposure value of the camera to be adjusted

Returns

See the Error Code

Remarks

- It is recommended to pre-allocate 21072 bytes for face verification/identification template data.
- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_GetTemplateQlt

Function Syntax

```
int __stdcall AMTNIRFace_GetTemplateQlt
        (
                void* context,
                unsigned char* verTemplate,
                int* score
                        );
```

Description

Gets the quality of the face verification/identification template (supports only the verification/identification template, and not the registration template generated by AMTNIRFace_MergeRegTemplate).

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
verTemplate	**In:** Face verification/identification template data
score	**Out:** Return the quality score of the corresponding face template (score range: 0 to 255)

Returns

See the Error Code

Remarks

- This interface is for reference only, there may be errors.
- Face quality score, the recommended threshold is: 50
- verTemplate can only be the verification/identification template data.

Click here to view the Function List.

AMTNIRFace_MergeRegTemplate

Function Syntax

```
int __stdcall AMTNIRFace_MergeRegTemplate
        (
                void* context,
                unsigned char*verTemplates,
                int mergedCount,
                unsigned char* pMergeTemplate,
                int* cbMergeTemplate
                        )
```

Description

Combines the 5-verification/identification template data into one registered template data.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
verTemplates	**In:** Verification/Identification template data (Supports 5 templates that are required to be merged into a one-dimensional array, it is recommended to pass only 5 verification/identification templates)
mergedCount	**Out:** Number of verification/identification template data (It is recommended to transfer 5 verification/identification templates, and only supports up to 5)
pMergeTemplate	**Out:** Registration template synthesized by multiple verification/identification templates (the generated template is used when AMTNIRFace_DBAdd is added)
cbMergeTemplate	**In:** pMergeTemplate memory allocation size
	Out: Returns the actual pMergeTemplate data length

Returns

See the Error Code

Remarks

- The face registration template suggests pre-allocating 28992 bytes.
- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_Verify

Function Syntax

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

Description

Performs the 1:1 face verification.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
regTemplate	**In:** Registration template data
verTemplate	**In:** Verification template data
score	**Out**: Returns the corresponding verification score

Returns

See the Error Code

Remarks

- Verification score range: 1~1000.
- Verification score threshold recommended value: 575.
- The interface score returns the corresponding verification score value, and the application layer determines the verification threshold.

Click here to view the Function List.

AMTNIRFace_DBVerifyByID

Function Syntax

```
int __stdcall AMTNIRFace_DBVerifyByID
        (
                void *context,
                const unsigned char*verTemplate,
                const char *faceID,
                int *score,
                bool IsAdapt,
                unsigned char*adaptFeature,
                int *cbAdaptFeature
                        );
```

Description

Performs the 1:1 verification with the specified faceID.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
verTemplate	**In:** Verification template data
faceID	**In:** The specified face ID
score	**Out**: Return the corresponding verification score
IsAdapt	**In:** Whether the registration template needs to be updated. true: means enable self–learning false: means disable self-learning
adaptFeature	**Out:** Return to the registration template after learning. It is recommended to pre-allocate 28992 bytes of memory (the returned registration template only needs to be updated to its own application database, and the algorithm is automatically updated to the 1:N bottom library)
cbAdaptFeature	**In:** Memory size allocated by adaptFeature (number of bytes)
	Out: Returns the actual length of the adaptFeature

Returns

See the Error Code

Example

```
int ret = -1;
int score = 0;
char szFaceID[256] = "faceid";
unsigned char *adaptTemplate = new unsigned char[28992];
memset(adaptTemplate,0,28992);
int cbAdaptTemplate = 28992;
ret = AMTNIRFace_DBVerifyByID(context,
verTemplate,szFaceID,&score,true,adaptTemplate,&cbAdaptTemplate);
if(adaptTemplate)
{
delete [] adaptTemplate;
adaptTemplate = NULL;
}
```

Remarks

- Verification score range: 1~1000.
- The recommended minimum score is 575.
- If the length of the self-learning registration template returned by cbAdaptFeature is equal to 0, then it means that the self-learning registration template is not successfully generated.
- If the returned length of the self-learning registration template is greater than 0, then it means that the self-learning registration template is successfully obtained and automatically updated to the 1:1 library.
- The interface score returns the corresponding verification score value, and the application layer determines the verification threshold.

Click here to view the Function List.

AMTNIRFace_DBAdd

Function Syntax

int __stdcall AMTNIRFace_DBAdd(void* context, char* faceID);

Description

Adds a registered template to the database.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
faceID	**In:** Face ID

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_DBDel

Function Syntax

int __stdcall AMTNIRFace_DBDel(void* context, char* faceID);

Description

Removes the specified face template from the database.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
faceID	**In:** Face ID

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_DBClear

Function Syntax

int __stdcall AMTNIRFace_DBClear(void* context);

Description

Clears the database.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_DBCount

Function Syntax

int __stdcall AMTNIRFace_DBCount(void* context, int* count);

Description

Gets the total number of face templates stored in the database.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
count	**Out:** Returns the total number of templates stored in the high-speed buffer

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface.

Click here to view the Function List.

AMTNIRFace_DBIdentify

Function Syntax

```
int __stdcall AMTNIRFace_DBIdentify
        (
                void* context,
                const unsigned char*verTemplate,
                char *faceID,
                int* score,
                bool IsAdapt,
                unsigned char*adaptFeature,
                int *cbAdaptFeature
                        );
```

Description

Performs the 1:N face identification.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
verTemplate	**In:** Identification templates
faceID	**Out:** Returns the face ID
score	**Out:** Returns the face identification score
IsAdapt	**In:** Whether the registration template needs to be updated. true: Means enable self–learning false: This means disable self - learning
adaptFeature	**Out**: Returns to the registration template after learning. It is recommended to pre-allocate 28992 bytes of memory (the returned registration template only needs to update to its own application database, and the algorithm is automatically updated to the 1:N library)
cbAdaptFeature	**In:** Memory size allocated by adaptFeature (number of bytes)
	Out: Returns the actual length of the adaptFeature

Returns

See the Error Code

Example

```
int ret = -1;
int score = 0;
char szFaceID[256] = {0};
unsigned char *adaptTemplate = new unsigned char[28992];
memset(adaptTemplate,0,28992);
int cbAdaptTemplate = 28992;
ret = AMTNIRFace_DBIdentify(context,
verTemplate,szFaceID,&score,true,adaptTemplate,&cbAdaptTemplate);
if(adaptTemplate)
{
delete [] adaptTemplate;
adaptTemplate = NULL;
}
```

Remarks

- Identification score range: 1~1000.
- The recommended minimum score is 585.
- This interface is a non-thread safe interface.
- If the length of the self-learning registration template returned by cbAdaptFeature is equal to 0 then it means that the self-learning registration template is not successfully generated.
- If the returned length of the self-learning registration template is greater than 0, then it means that the self-learning registration template is successfully obtained and automatically updated to the 1:N Bottom library.
- The interface score returns the corresponding identification score value, and the application layer determines the identification threshold.

Click here to view the Function List.

AMTNIRFace_GetFacePosition

Function Syntax

```
int __stdcall AMTNIRFace_GetFacePosition
        (
                void* context,
                int *positions,
                int count
                        );
```

Description

Gets the position coordinates of the near-infrared face.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
positions	**Out:** Face coordinates
count	**In:** positions memory allocation size (it is recommended to allocate 12 int data)

Returns

See the Error Code

Remarks

- positions return value description:
- positions[0]~positions[7]Four coordinate points of the rectangular frame of the near-infrared face: p0.x p0.y p1.x p1.y p2.x p2.y p3.x p3.y. (The coordinates of the upper left corner of the rectangular frame of the face are arranged clockwise).

positions[8]	X coordinate of the left eye
positions[9]	Y coordinate of the left eye
positions[10]	X coordinate of the right eye
positions[11]	Y coordinate of the right eye

Click here to view the Function List.

AMTNIRFace_DetectAndGetPos

Function Syntax

```
int __stdcall AMTNIRFace_DetectAndGetPos
        (
                void* context,
                unsigned char*grayIr,
                unsigned char* bgrColor,
                int width,
                int height,
                int *yaws,
                int *pitchs,
                int*rolls,
                int *points,
                int len
)
```

Description

Face detection and face position acquisition.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
grayIr	**In:** Original image data with 8-bit grayscale image bit depth.
bgrColor	**In:** Original image data with a 24-bit BGR image bit depth
width	**In:** Image width
height	**In:** Image height
yaws	**Out:** yaws[0]: is the infrared face yaw value yaws[1]: is the visible light face yaw value (allocate 2 arrays of int type length)
pitchs	**Out:** pitchs[0]: is the infrared face pitch value pitchs[1]: is the visible light face pitch value (allocating 2 arrays of int type length)
rolls	**Out:**

	rolls[0] is the infrared face roll value rolls[1] is the visible light face roll value (allocate 2 arrays of int type length)
points	**Out:** points[0]~points[7] Four coordinate points of the rectangular frame of near infrared face points[8]~points[15] Four coordinate points of the rectangular frame of near infrared face The four coordinate points p0.x p0.y p1.x p1.y p2.x p2.y p3.x p3.y of the rectangular frame of the face are arranged in order (from the coordinates of the upper left corner of the rectangular frame of the face are arranged clockwise)
len	**In:** Points array size, allocate 16 arrays of int type length

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface

Click here to view the Function List.

AMTNIRFace_GetLiveness

Function Syntax

```
int __stdcall AMTNIRFace_GetLiveness
        (
                void* context,
                unsigned char*grayIr,
                unsigned char* bgrColor,
                int width,
                    int height,
                    float* fScores
)
```

Description

Face live detection.

Parameters

Parameter	Description
context	**In:** Algorithm face instance pointer
grayIr	**In:** Original image data with 8-bit grayscale image bit depth.
bgrColor	**In:** Original image data with a 24-bit BGR image bit depth.
width	**In:** Image width
height	**In:** Image height
fScores	**Out:** Liveness score When the binocular is alive (fScores[0] is the infrared live value; fScores[1] is the visible light live value)

Returns

See the Error Code

Remarks

- This interface is a non-thread safe interface.
- You must call the AMTNIRFace_DetectAndGetPos interface before calling this interface.
- Recommended liveness threshold is 0.7

Click here to view the Function List.

Appendix

Appendix 1: Error Code

Error Code	Description
0	Successful operation
-1	Image size conversion error, face detection failure
-3	No face detected
-5	Failed to synthesize registration template
-8	Algorithm library memory allocation error
-15	Feature extraction failed
-103	No such faceid in the database (no such faceid in the cache)
-105	The feature of the faceid in the database is invalid (in the high-speed buffer)
-106	Duplicate added faceid
-200	Database is full (cache area)
-1000	Dongle error
-1001	Algorithm library initialization failed
-1002	Algorithm library is not initialized
-1003	Invalid handle
-1004	Null pointer
-1005	The interface is not supported
-1006	Invalid parameter
-1007	Face detection failed during live detection
-1008	Not enough memory allocated
-1012	The face index is invalid
-1015	Failed to allocate memory
-1020	Failed to load algorithm library
-1021	Failed to initialize visible light face detection engine
-1023	Failed to initialize visible light live detection engine
-1024	The algorithm did not detect the near-infrared face before the live detection
-1025	The algorithm did not detect the visible light face before the live detection

Appendix 2: Glossary

The following definitions will help you understand basic functions of a near-infrared face recognition application and complete integrated development of such an application.

Verification/Identification template

Verification/Identification templates are used for 1:1 or 1:N face verification/identification or merged into a registered template for face registration.

1:1 face verification

1:1 face verification, also called face verification, is a process of verifying whether a user has a valid identity based on the user ID and face template or determining whether a registered template and several verification templates are extracted from the same face.

1:N face identification

1:N face identification, also called face recognition, is a process of determining whether a user exists in the system based on the face of the user, without the user ID. Specifically, the application looks up the database of registered face templates based on the input face template and returns the name of the user meeting the threshold, face similarity degree, and other related information.

Registered template

The face registration template returned by AMTNIRFace_MergeRegTemplate, or the self-learning registration template returned by AMTNIRFace_DBIdentify and AMTNIRFace_DBVerifyByID.

Registered face

The face recognition module/collector captures five face images of the same user to extract verification/identification templates, merges the verification/identification templates into a registered template, and then loads it to the backend database as a registered face for subsequent face recognition.

190 Bluegrass Valley Pkwy,

Alpharetta, GA 30005, USA

E-mail: info@armatura.us

www.armatura.us