API Development Manual

Armatura FacePro Windows SDK

Date: Jan 2023

SDK Version: 5.8

Document Version: 1.1

English Version

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

Copyright © 2023 Armatura LLC. All rights reserved.

Trademark

Disclaimer

The ARMATURA product manual serves as a comprehensive guide for operating and maintaining the ARMATURA product. The manual and all accompanying documents, drawings, tables, etc. are protected by copyright and are the property of ARMATURA. It is strictly prohibited to use or share the contents of this manual with any third party without obtaining express written permission from ARMATURA.

To ensure optimal performance and safety, it is crucial to read and fully understand the entire manual before beginning operation and maintenance of the product. If any aspect of the manual seems unclear or incomplete, please contact ARMATURA for clarification.

Proper training and familiarity with the product's design are essential prerequisites for successful operation and maintenance. Additionally, it is crucial to read, understand, and strictly follow the safety instructions outlined in the manual.

In the event of any discrepancies between the terms and conditions outlined in this manual and those specified in the contract or other contract-related documents, the contract conditions shall take precedence.

ARMATURA does not provide any warranties, guarantees, or representations regarding the completeness or accuracy of the information contained in this manual or any subsequent amendments. ARMATURA also does not assume responsibility for any errors or omissions and the user assumes all risks associated with the use of the information provided.

ARMATURA shall not be liable for any incidental, consequential, indirect, special, or exemplary damages arising from or in connection with the use of the information contained in this manual.

This manual may contain technical inaccuracies or typographical errors and is subject to periodic updates and revisions. ARMATURA reserves the right to add, delete, amend, or modify the information contained in the manual at any time and such amendments shall not give any right to claim any compensation or damages under any circumstances.

ARMATURA shall not be held responsible for any malfunctions or damage resulting from non-compliance with the instructions outlined in the manual or operation beyond prescribed limits or conditions.

The product may be updated periodically without prior notice and the latest operation procedures and relevant documents can be found at **https://www.armatura.us**.

If you have any issues related to the product, please do not hesitate to contact us.

Armatura Office

Address	190 Bluegrass Valley Pkwy,
	Alpharetta, Georgia, 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 cutting-edge biometric solution provider that stays at the forefront of technology by continuously researching and developing new hardware designs, algorithms, and software. As a leader in the industry, ARMATURA holds a vast portfolio of patents in biometric recognition technology. Our products are designed for businesses that demand the highest level of security, accuracy, and speed in user identification.

Our biometric hardware and software are integrated into some of the most reliable and well-known brands in workforce management (WFM) terminals, Point-of-Sale (PoS) terminals, intercoms, electronic safes, metal key lockers, dangerous machinery, and many other products that require secure and accurate user authentication. Trust ARMATURA to provide you with the most advanced and reliable biometric solutions for your business needs.

About the Manual

This manual is intended to provide a comprehensive guide to the use of Armatura FacePro 5.8 SDK for Windows. It covers all the necessary information to help users understand and operate the software effectively.

Please note that the figures and images provided in this manual are for illustrative purposes only and may not reflect the exact appearance of the actual products. Additionally, the software and its features are subject to change and may not be exactly as depicted in the manual. The manual is intended to be used as a reference guide and should be read in conjunction with the SDK documentation provided by ARMATURA.

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.
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 Overview

Utilizing the advanced facial recognition algorithms, Armtura FacePro SDK empowers developers to easily integrate biometric-based applications with ease. Our comprehensive SDK guide provides developers with all the necessary information to build and integrate face recognition seamlessly. With its user-friendly development kit and detailed function specifications, Armatura FacePro 5.8 SDK is the perfect choice for any biometric integration project which requires the facial recognition-based identification and authentication features. It ensures to make your development process easier and more efficient with Armatura FacePro SDK..

1.1 About Armatura FacePro 5.8 Algorithm

Armatura FacePro 5.8 algorithm is a cutting-edge, visible light-based facial recognition solution that utilizes advanced deep-learning techniques to accurately detect key facial features and landmarks, such as eyes, lips, nose tips, and contour elements. This powerful algorithm supports a wide range of features, including face detection, liveness detection, mask detection, age estimation, gender identification, and facial matching.

Designed to provide strong adaptability to complex application environments, the Armatura FacePro 5.8 algorithm is able to withstand challenges such as hair accessories occlusion, image blurring, and varying lighting conditions. Additionally, the algorithm is user-friendly for software developers and integrators, allowing for easy customization to meet specific business and customer requirements.

Armatura FacePro SDK is built on deep-learning trained face models, which may vary between different SDK versions and platforms. As a result, the face template created for identification and verification may not be consistent across different versions and platforms, meaning that the face template is not transferable between SDK versions and platforms. This ensures that the highest level of security and accuracy is maintained for all users.

1.2 Features

1:N Identification

Armatura FacePro algorithm utilizes stable facial features and employs a multi-level identification method for optimal classifier parameterization, providing a robust means of multi-factor identification for large user populations..

Analysis of face attributes

Armatura FacePro algorithm utilizes advanced computer vision techniques and deep-learning technology, trained on vast amounts of data, to provide comprehensive facial attribute analysis. This includes the detection of gender, age, facial expression, and mask usage for the target individual..

Face posture

Armatura FacePro algorithm exhibits robust posture adaptability, with the ability to accurately identify individuals despite variations in head tilt, rotation, and yaw. This is achieved through advanced image processing techniques such as affine transformation and feature-based alignment, which enable the system to normalize and compensate for changes in facial posture.

The FacePro algorithm has the capability to accurately identify individuals within a range of yaw angles up to 30 degrees and pitch angles up to 25 degrees, providing comprehensive coverage for a wide range of practical applications.

Face expression

Armatura FacePro algorithm has been specifically designed to support accurate facial recognition, even under unusual or unnatural facial expressions. Examples of such expressions include laughing (with teeth or mouth visible), raised eyebrows, closed eyes, and frowning eyebrows. Despite these challenging conditions, the algorithm is able to deliver high-precision results.

High performance

Utilizing the deep-learning algorithm, Armatura FacePro algorithm is able to perform face detection and identification from single frame image at millisecond level. It makes possible to process the real-time face recognition on video streams captured from video surveillance devices or smart devices with digital cameras.

Multiple person tracking

Trained by deep-learning model with cutting-edge multiple face tracking technology, Armatura FacePro allows for simultaneous identification and analysis of multiple individuals on a single frame image captured from video stream. This powerful tool can accurately detect and track multiple faces in real-time, meanwhile it also provides valuable analytic information of facial attributes and expressions.

2 Technical Specifications

Development Language

The FacePro SDK is built on standard Win32 API and supports C, C++, and C# language programming.

Platform Requirements

The FacePro SDK is compatible with 32-bit and 64-bit versions of Windows XP SP3 or higher operating system.

Technical Specification

Parameter	Description
Template size	2048 bytes
Face Posture adaptability	Yaw ≤ 30°, Pitch ≤ 30°, Roll ≤ 30°
1:N Capacity	50,000
Face detection	< 50 ms
Face feature extraction	< 350 ms
Face Identification (1:50,000)	< 100 ms
Accuracy	FAR = 0.03% when FRR = 1.86%

Note:

The performance metrics for the algorithm are derived from a proprietary face image dataset with a resolution of 640x640, running a computer system equipped with 8GB of memory and a quad-core Inter(R) Core(TM) i5-3210M CPU @2.5GHz processor.

3 SDK Installation

3.1 Deploy SDK File

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

Model Name	Library File Name
Algorithm model file	sdk_x86_face_attr_fp_v1.0.1.bin
	sdk_x86_face_hvdet_fp_large_v1.0.3.bin
	sdk_x86_feature_Lumia_fp_v1.0.1.bin
	sdk_x86_liveness_fp_v1.0.1.bin
Algorithm core library	libbaselayer.dll
	libmidlayer.dll
	libsdkface.dll
	libsdksearch.dll
Third-party tool library	pthreadVC2.dll
	turbojpeg.dll
Algorithm core library	sdklayer1.dll
Dynamic link library of visible light face API	liveface.dll

3.2 Project Configuration

You can copy the FacePro SDK DLL files directly to your development and deployment environment without extra installation steps.

Before deploying FacePro SDK package, please make sure that your operating system, computer configuration, or Windows mobile terminal device meets the system requirement。

Next copy the following FacePro SDK DLL files: libbaselayer.dll, libmidlayer.dll, libsdkface.dll, libsdksearch.dll, meglayer1.dll, pthreadVC2.dll, turbojpeg.dll, sdklayer1.dll, liveface.dll to the specified directory to build your application.

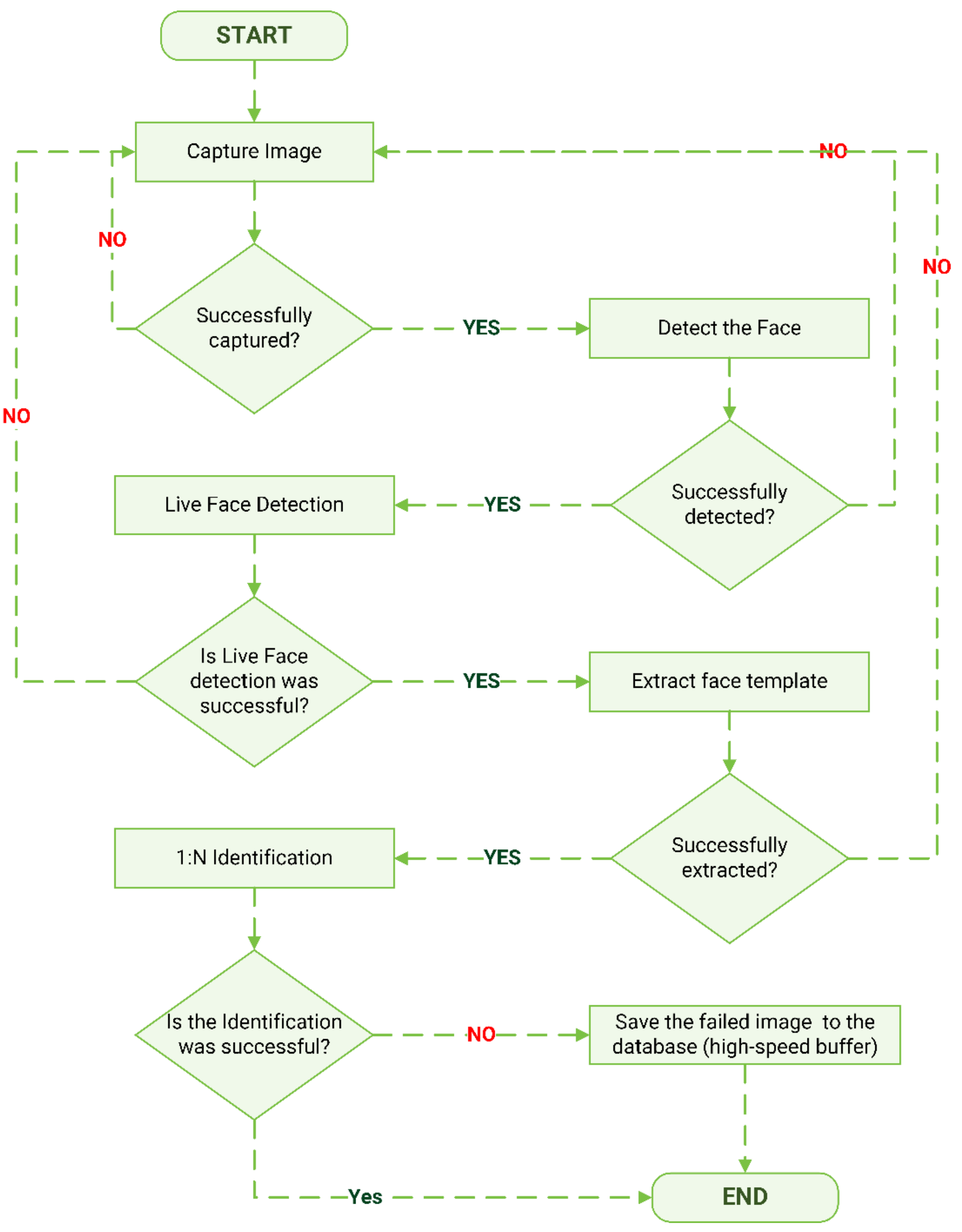
4 Programming Guide

The following guide will explain the face recognition operation workflow and provide the development reference to developers to understand the registration/identification/mask detection workflow and procedure implemented by the FacePro 5.8 algorithm.

4.1.1 Registration Procedure

When registering an individual face, the SDK can directly take the extracted face template as a registration template. For more information on this process and its specific implementation, please refer to the API specification provided in this document.

Registration Process Flow



Process Description

2) The program starts to capture the face images.

3) `DetectFaces` is called to detect the face after the face image is captured successfully.

4) `PredictLiveness` is called to perform liveness detection after the face is detected successfully.

5) `ExtractFeature` is called to extract the face features and create template upon positive liveliness detection.

6) `Search` is called to perform 1:N matching the candidate template to these in the

database and check if the same face has been enrolled or not. This step is called deduplication as well.

7) If there is successful match found in the database, the face has been registered and the application can take proper action to handle duplicated case.

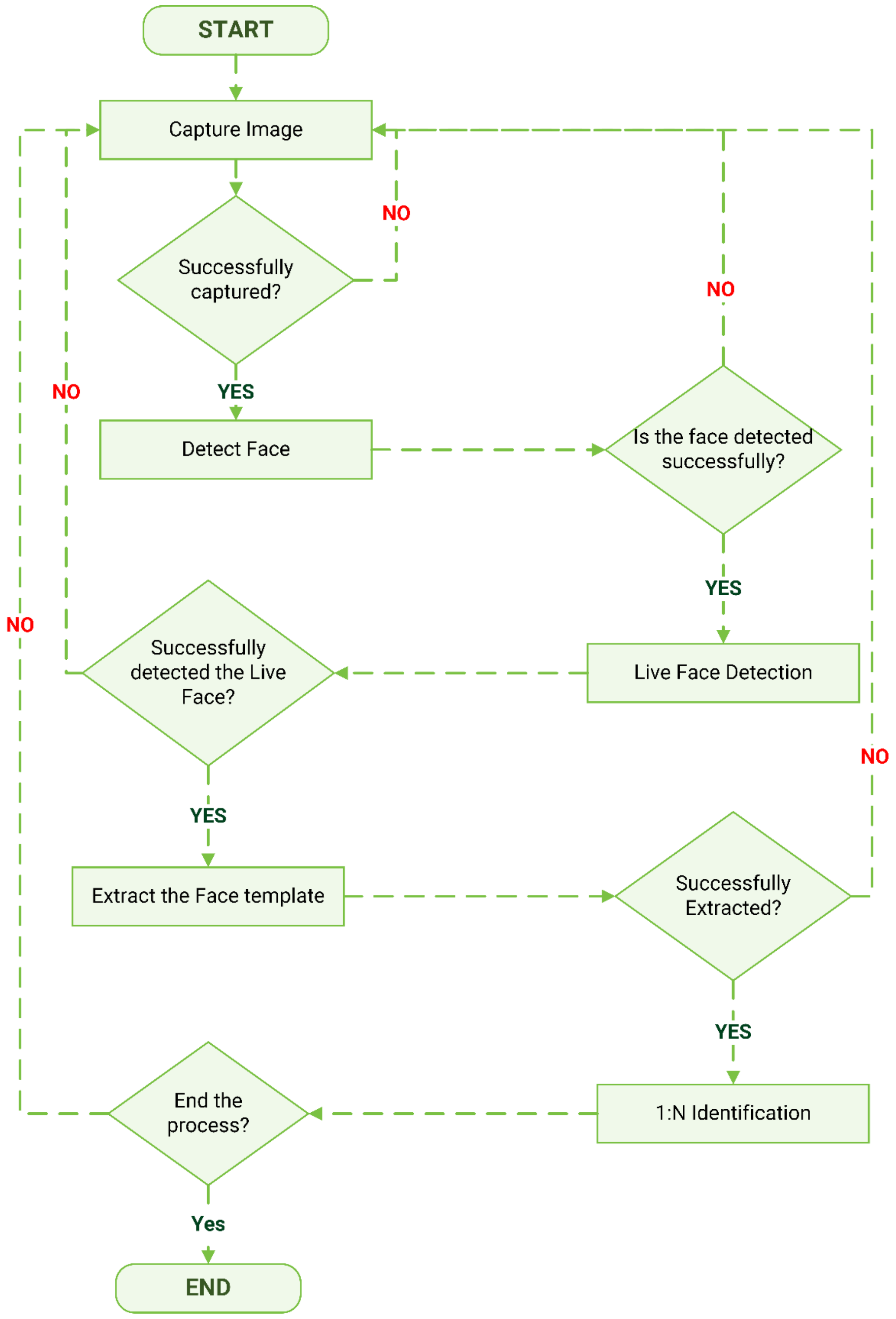
8) If no match found in the database, **`InsertFaceToGroup`** is called to add the candidate face template to the in-memory library (or high-speed cache) for runtime matching operation and the same face template is saved into to the database for persistence.

9) The registration process is completed and stops.

4.1.2 1:N Identification Process

In order to perform 1:N identification, it is required that all enrolled templates shall be loaded from the database to the in-memory library (high-speed cache) before performing 1:N identification. In memory matching process avoids the disk I/O latency and is speedy. The algorithm library is initiated and InsertFaceToGroup is called to add all the enrolled templates to the in-memory library.

Identification Process Flow



Process Description

1) The program starts to capture the face images.
2) `DetectFaces` is called to detect the face after the image is captured successfully.
3) `PredictLiveness` is called to perform liveliness detection after the face is successfully detected on the image.

4) `ExtractFeature` is called to extract the face features and create candiate template upon positive liveliness detection.

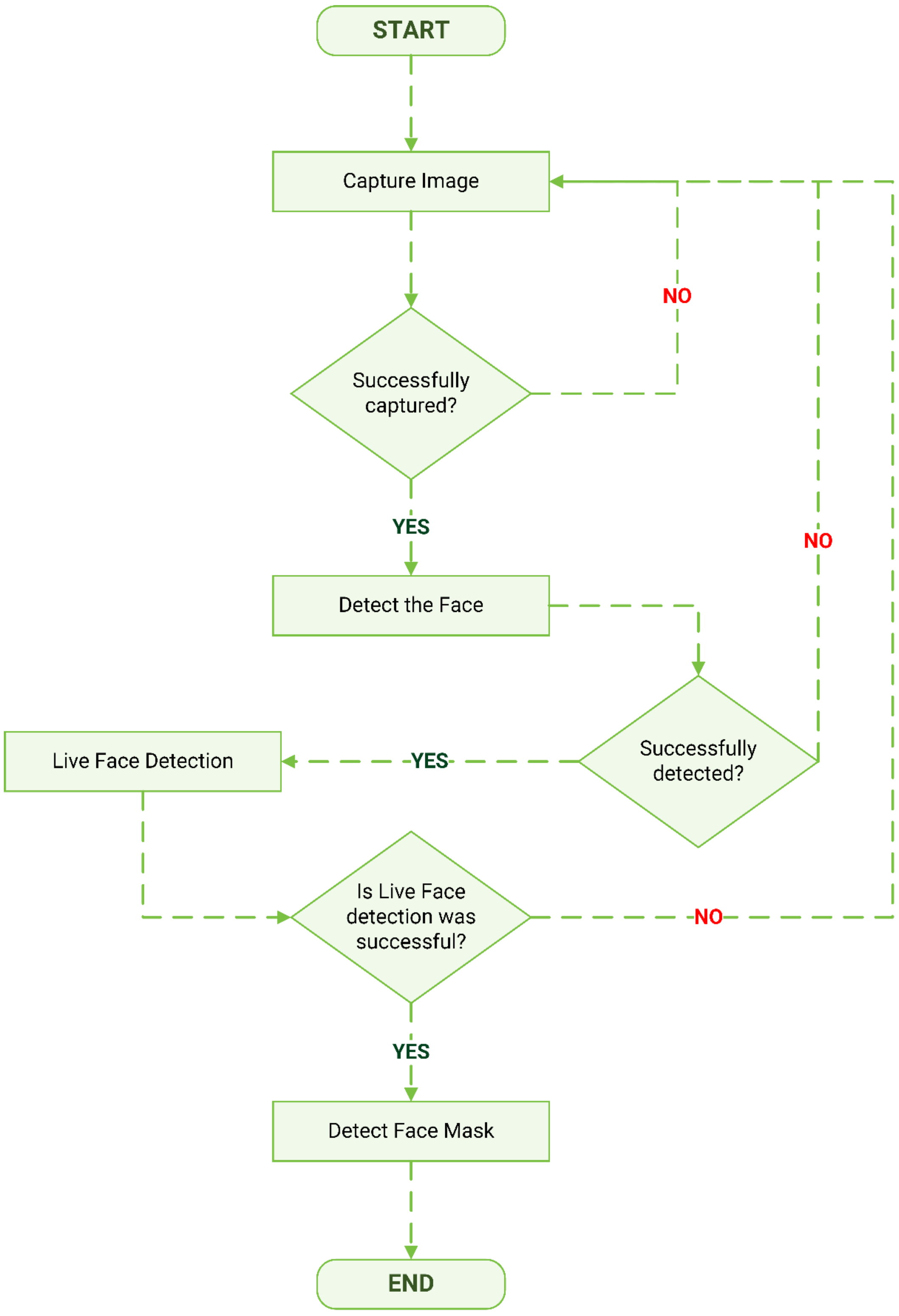
5) `Search` is called to perform 1:N identification and returns the matching result after the candidate template is successfully created.

6) 1:N identification process is completed and stops here.

4.1.3 Mask Detection

Facial mask detection can be performed directly after the face is detected on the target image. For more information on this process and its specific implementation, please refer to the API description provided.

Mask Detection Process Flow



Process Description

1) The program starts to capture the face images.

2) `DetectFaces` is called to detect faces after the image is captured successfully.

3) `PredictLiveness` is called to perform liveliness detection after the face is

successfully detected on the image.

4) **`PredictAttribute`** is called to perform mask detection upon positive liveness detection.

5) Face mask detection operation is completed and stops here.

5 SDK Interface Description

5.1 Visible Light Face Template Format

Template Type	Data Length	Description
Face template	2048 bytes	Templates are used as registration templates and identification templates.

5.2 Visible Light Face API Description

5.2.1 LiveFace.dll

Function List

Interface	Description
Version	Gets the SDK version number
Init	Initializes algorithm resources
Terminate	Releases algorithm resources
FreeMemory	Releases the memory
CreateDetectHandle	Creates the face detection instance handle.
DetectFaces	Detects the face.
DestroyDetectHandle	Releases the face detection instance handle
CreateFeatureHandle	Creates an instance handle for extracting the face template
ExtractFeature	Extracts the face template
DestroyFeatureHandle	Releases the face template instance handle
CreateLivenessHandle	Creates an instance handle for alive (liveliness) face detection
PredictLiveness	Detects the alive faces (liveliness of the face)
DestroyLivenessHandle	Releases the instance handle of alive (liveliness) face detection
CreateAttributeHandle	Creates the instance handle for face attribute detection
PredictAttribute	Detects the face attributes
DestroyAttributeHandle	Releases the instance handle of face attribute detection
CreateCompareHandle	Creates the instance handle for 1:1 face verification
Compare	Performs the 1:1 verification process
DestroyCompareHandle	Releases the 1:1 face instance handle
CreateSearchHandle	Creates a 1:N high-speed cache
CreateGroup	Creates a group in 1:N high-speed cache
InsertFaceToGroup	Adds face templates to a group specified by the 1:N high-speed cache
DeleteFaceFromGroup	Deletes the face template in the group specified by the 1:N high-speed cache
Search	Performs 1:N recognition in the specified group ID of 1:N high-

	speed cache
DestroyGroup	Deletes the specified group in 1:N high-speed cache
DestroySearchHandle	Releases 1:N high-speed cache resources
AnalyzeDetectResult	Analyzes the face information structure
AnalyzeFeatureResult	Analyzes the face template structure
AnalyzeLivenessResult	Analyzes the face liveness feature structure
AnalyzeFaceAttributeResult	Analyzes the face attribute structure
ConverBioFeatureToStandFeature	Converts the original template of the algorithm into integrated template
ConverStandFeatureToBioFeature	Converts the integrated template into algorithm original template

Data Structure Description

Function Syntax

```
typedef struct _FaceDetectConfig{
            int face_min;
            float pose_roll_upper_threshold;
            float pose_yaw_upper_threshold;
            float pose_pitch_upper_threshold;
            float blurriness_upper_threshold;
            int brightness_low_threshold;
            int brightness_upper_threshold;
            int brightness_deviation_threshold;
            float face_completeness_threshold;
            int reserved[23];
      } TFaceDetectConfig,*PFaceDetectConfig;

typedef struct _AttributeResult // Click here to view its description
      {
            int magic;
            int gender;
            int age;
            int maskStatus;
      }TAttributeResult,*PAttributeResult;

typedef struct _FeatureResult // Click here to view its description
      {
            char* featureData;
            int featureLength;
      }TFeatureResult,*PFeatureResult;

typedef struct _LivenessResult // Click here to view its description
      {
            float livenessScore;
```

```
                int reserved[8];
        }TLivenessResult,*PLivenessResult;

typedef struct _FaceLandmark // Click here to view its description
        {
                char* landmark_data;
                int landmark_length;
        } TFaceLandmark,*PFaceLandmark;

typedef struct _FaceRect // Click here to view its description
        {
                int left;
                int top;
                int right;
                int bottom;
        } TFaceRect,*PFaceRect;

typedef struct _FacePose // Click here to view its description
        {
                float roll;
                float pitch;
                float yaw;
        } TFacePose,*PFacePose;
typedef struct _DetectFaceInfo // Click here to view its description
        {
                FaceRect rect;
                FaceRect extent_rect;
                FacePose pose;
                float blur;
                TFaceLandmark face_landmark;
                int brightness;
                int brightness_deviation;
                int reserved[24];
        } TDetectFaceInfo, *PDetectFaceInfo;

typedef struct _DetectFaceResult // Click here to view its description
        {
                int magic;
                PDetectFaceInfo face_info;
                int face_count;
                int reserved[24];
        } TDetectFaceResult, *PDetectFaceResult;

typedef struct _IdentifyResult // Click here to view its description
        {
                float scores;
                unsigned int face_id;
            } TIdentifyFaceResult, *PIdentifyFaceResult;
```

FaceDetectConfig

Face detection configuration parameters

Parameters

Parameter	Description
face_min	The minimum pixel to capture a face (that is, it ignores longer distance faces.) • The range is 0 - maximum resolution. • The recommended value is 50.
pose_roll_upper_threshold	Roll threshold sets the angle constraint of the nose-center rotation for the captured face (that is, it ignores the larger angle of the faces during identification process). • The range is 0-180 degrees. • The recommended value is 30.
pose_yaw_upper_threshold	It is to set the angle constraint of the left and right deflection angles for the captured face (that is, it ignores the larger angle of the faces). • The range is 0-180 degrees. • The recommended value is 30.
pose_pitch_upper_threshold	It is to set the angle constraint of the up and down pitch angles for the captured face (that is, it ignores the larger angle of the faces). • The range is 0-180 degrees. • The recommended value is 30.
blurriness_upper_threshold	It is to set the constraint of the blur degree for the captured face. • The range is (0-1). • The recommended value is 0.7.
brightness_low_threshold	It is to set the lower limit for the face brightness. • The range is (0-255). • The recommended value is 70; 0 for no limit.
brightness_upper_threshold	It is to set the Upper limit for face brightness. • The range is (0-255). • The recommended value is 210; 0 for no limit.
brightness_deviation_threshold	It is to set the threshold of the standard deviation for the face brightness, such as a face with sunglasses. If the standard deviation for the brightness is larger, then the face quality may be poor. • The range is (0-255). • The recommended value is 60; 0 for no limit.
face_completeness_threshold	Showing face integrity data for collected image. • The range is (0-1). • The recommended value is 0.9; when it is set to 0.0. • The SDK uses the internal default value 0.9.
reserved	Reserved
TFaceDetectConfig	Structure of configuration parameter for Face detection

PFaceDetectConfig	Pointer of face detection configuring parameter structure

AttributeResult

Parameters

Parameter	Description
gender	0 Unknown 1 Male 2 Female
age	Age
maskStatus	0 Unknown 1 Without mask 2 With mask
TAttributeResult	Face attribute structure
PAttributeResult	Face attribute structure pointer

FeatureResult

Parameters

Parameter	Description
featureData	Face template
featureLength	Face template length
TFeatureResult	Face template feature structure
PFeatureResult	Face template feature structure pointer

LivenessResult

Alive face detection result

Parameters

Parameter	Description
livenessScore	Liveliness face value (0-1)
reserved[8]	Reserved parameters
TLivenessResult	Alive face detection result structure
PLivenessResult	Alive face detection result structure pointer

FaceLandmark

Parameters

Parameter	Description
landmark_data	Face landmark data
landmark_length	Face landmark data length
TFaceLandmark	Face landmark structure

PFaceLandmark	Face landmark structure pointer

FaceRect

Face rectangle coordinates (upper left corner coordinates and right corner coordinates).

Parameters

Parameter	Description
left	X axis of the upper left coordinate of the face rectangle
top	Y axis of the upper left coordinate of the face rectangle
right	X axis of the lower right coordinate of the face rectangle
bottom	Y axis of the lower right coordinate of the face rectangle
TFaceRect	Face rectangle coordinate structure
PFaceRect	Face rectangle coordinate structure pointer

FacePose

Parameters

Parameter	Description
roll	The tilt angle for the captured face image (rotation around the Z-axis): ±90
pitch	Moving the head up and down (rotation around the X-axis): ±90
yaw	The angle of moving the head left and right (rotation around the Y-axis): ±90
TFacePose	Face pose structure
PFacePose	Face pose structure pointer

DetectFaceInfo

Parameters

Parameter	Description
rect	The coordinates of detected face frame
extent_rect	Expanding the coordinates to include the entire face, which can be used to crop the image with the whole detected face.
pose	Face angle attributes
blur	The blur degree attribute of the face (0-1) 0 — for the clearest 1 — for the blurriest
face_landmark	Face coordinate information
brightness	Face brightness
brightness_deviation	Standard deviation of face brightness
reserved[24]	Reserved
TDetectFaceInfo	Face information structure
PDetectFaceInfo	Face information structure pointer

DetectFaceResult

Parameters

Parameter	Description
face_info	List of captured face information
face_count	The total number of faces detected
reserved[24]	Reserved
TDetectFaceResult	Face data structure
PDetectFaceResult	Face data structure pointer

IdentifyResult

Parameters

Parameter	Description
scores	The scores matched by the face in the low-level library (high-speed cache)
face_id	Face id matched by the face in the low-level library (high-speed cache)
TIdentifyFaceResult	1:N recognition result structure
PIdentifyFaceResult	1:N recognition result structure pointer

Version

Function Syntax

```
    int __stdcall Version
        (
                char* version,
                int* size
        );
```

Description

Gets the SDK version number.

Parameter

Parameter	**Description**
version	**Out**: Returns the version number (it is recommended to pre-allocate 128 bytes)
size	**In**: Memory size (bytes) allocated for the version
	Out: Returns the actual version length

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Example

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

Remarks

- Click here to view the Function List.

Init

Function Syntax

```
int __stdcall Init();
```

Description

Initializes the algorithm resources.

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Call Init before calling other APIs, and other APIs can be used normally after calling this API successfully.
- In the entire program cycle, Init only needs to be initialized once.
- Click here to view the Function List.

Terminate

Function Syntax

```
int __stdcall Terminate();
```

Description

Releases the algorithm resources

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Call this API at the end of the program.
- Click here to view the Function List.

FreeMemory

Function Syntax

```
int __stdcall FreeMemory(void *pResult);
```

Description

Releases the memory.

Parameter

Parameter	Description
pResult	**In**: Pointer to release the memory.

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

CreateDetectHandle

Function Syntax

```
int __stdcall CreateDetectHandle(void **detectHandle);
```

Description

Creates the face detection instance handle.

Parameter

Parameter	Description
detectHandle	**Out**: Face detection instance handle

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- You can create multiple face detection handles.
- Click here to view the Function List.

DetectFaces

Function Syntax

```
int __stdcall DetectFaces
    (
            void* detectHandle,
            TFaceDetectConfig detectConfig,
            unsigned char* rawImage,
            int width,
            int height,
            int* detectedFaces,
            PDetectFaceResult *detectResult
    );
```

Description
Detects the face.

Parameter

Parameter	Description
detectHandle	**In**: Face detection instance handle
detectConfig	**In**: Face detection configuration parameters (see Structure Description)
rawImage	**In**: BGR image bit depth of the original image data in 24 bits
width	**In**: Image width
height	**In**: Image height
detectedFaces	**Out**: Number of detected faces
detectResult	**Out**: Detected face data (used to get face attributes, detect alive faces, and extract face templates). It needs to be released by calling FreeMemory after use.

Returns
Returns the Error Code. See the Appendix 1 for error code details.

Remarks
- After DetectResult is used, it needs to be released by calling the FreeMemory API.
- For related structures, see Structure Description.
- Click here to view the Function List.

DestroyDetectHandle

Function Syntax

```
    int __stdcall DestroyDetectHandle(void *detectHandle);
```

Description
Releases face detection instance handle.

Parameter

Parameter	Description
detectHandle	**In**: Face detection instance handle

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- If the face detection instance handle is no longer needed to be used, it is necessary to call this API to release the handle.
- Click here to view the Function List.

CreateFeatureHandle

Function Syntax

```
int __stdcall CreateFeaturetHandle(void **featureHandle);
```

Description

Creates instance handle for extracting face template.

Parameter

Parameter	Description
featureHandle	**In**: Face template instance handle

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- It is possible to create multiple handles for extracting face templates.
- Click here to view the Function List.

ExtractFeature

Function Syntax

```
int __stdcall ExtractFeature
    (
            void* featureHandle,
            PDetectFaceResult detectResult,
            unsigned char* rawImage,
            int width,
            int height,
            PFeatureResult *featureResults
    );
```

Description

Extracts the face template.

Parameter

Parameter	Description

featureHandle	**In**: Face template instance handle
detectResult	**In**: Face data instance handle (see DetectFaces API description)
rawImage	**In**: BGR image bit depth is the original image data in 24 bits
width	**In**: Image width
height	**In**: Image height
featureResults	**Out**: The extracted face template data needs to be released by calling FreeMemory after use.

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After using featureResults, it is necessary to call FreeMemory API to release it.
- This API needs to be called after DetectFaces is successfully called.
- For related structures, see Structure Description
- Click here to view the Function List.

DestroyFeatureHandle

Function Syntax

```
int __stdcall DestroyFeatureHandle(void *featureHandle);
```

Description

Releases the face template instance handle.

Parameter

Parameter	Description
featureHandle	**In**: Face template instance handle

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- If you no longer need to use the face template instance handle, you need to call this API to release it
- Click here to view the Function List.

CreateLivenessHandle

Function Syntax

```
int __stdcall CreateLivenessHandle(void **livenessHandle);
```

Description

Creates an instance handle of alive face (i.e., liveliness) detection.

Parameter

Parameter	Description
livenessHandle	**Out**: Instance handle of alive face detection

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- You can create multiple instance handles of alive face detection
- Click here to view the Function List.

PredictLiveness

Function Syntax

```
int __stdcall PredictLiveness
    (
            void* livenessHandle,
            PDetectFaceResult detectResult,
            unsigned char* rawImage,
            int width,
            int height,
            PLivenessResult *livenessResults
    );
```

Description

Detects alive faces.

Parameter

Parameter	Description
livenessHandle	**In**: Instance handle of alive face detection
detectResult	**In**: Face data instance handle (see DetectFaces API description)
rawImage	**In**: BGR image bit depth, which refers to the original image data in 24 bits
width	**In**: Image width
height	**In**: Image height
livenessResults	**Out**: The detected alive face data should be released after calling FreeMemory function

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After using livenessResults, you need to call the FreeMemory API to release it.

- This API needs to be called after DetectFaces is successfully called.
- For related structures, see Structure Description
- Click here to view the Function List.

DestroyLivenessHandle

Function Syntax

```
int __stdcall DestroyLivenessHandle(void *livenessHandle);
```

Description

Releases the instance handle of alive face detection.

Parameter

Parameter	Description
livenessHandle	**Out**: Instance handle of alive face detection

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- If you no longer need to use the instance handle of alive face detection, you need to call this API to release it.
- Click here to view the Function List.

CreateAttributeHandle

Function Syntax

```
int __stdcall CreateAttributeHandle(void **attributeHandle);
```

Description

Creates the instance handle for face attribute detection.

Parameter

Parameter	Description
attributeHandle	**Out**: Instance handle of face attribute detection

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- You can create multiple instance handles of live face detection.
- Click here to view the Function List.

PredictAttribute

Function Syntax

```
int __stdcall PredictAttribute
    (
            void* attributeHandle,
            PDetectFaceResult detectResult,
            unsigned char* rawImage,
            int width,
            int height,
            PAttributeResult *attributeResults
    );
```

Description

Detects the face attributes.

Parameter

Parameter	Description
attributeHandle	**In**: Instance handle of face attribute detection
detectResult	**In**: Face data instance handle (see DetectFaces API description)
rawImage	**In**: BGR image bit depth is the original image data in 24 bits
width	**In**: Image width
height	**In**: Image height
attributeResults	**Out**: The detected face attribute data needs to be released after calling FreeMemory.

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After using attributeResults, you need to call the FreeMemory API to release it.
- This API needs to be called after DetectFaces is successfully called.
- For related structures, see Structure Description.
- Click here to view the Function List.

DestroyAttributeHandle

Function Syntax

```
int __stdcall DestroyAttributeHandle(void *attributeHandle);
```

Description
Releases the instance handle of the face attribute detection.

Parameter

Parameter	Description
attributeHandle	**Out**: Instance handle of face attribute detection

Returns
Returns the Error Code. See the Appendix 1 for error code details.

Remarks
- For the instance handle of face attribute detection that is no longer needed, it is required to call this API to release it.
- Click here to view the Function List.

CreateCompareHandle

Function Syntax

```
int __stdcall CreateCompareHandle(void **compareHandle);
```

Description
Creates the instance handle of face 1: 1 verification.

Parameter

Parameter	Description
compareHandle	**Out**: Instance handle of face 1: 1 verification

Returns
Returns the Error Code. See the Appendix 1 for error code details.

Remarks
- You can create multiple 1: 1 face instance handles.
- Click here to view the Function List.

Compare

Function Syntax

```
int __stdcall Compare
    (
        void* compareHandle,
        const char *firstFeature,
        int cbFirstFeature,
        const char *secondFeature,
        int cbSecondFeature,
        float *score
    );
```

Description
Performs the 1:1 face verification.

Parameter

Parameter	Description
compareHandle	**In**: Instance handle of face 1: 1 verification
firstFeature	**In**: Face template data 1
cbFirstFeature	**In**: Length of the face template 1
secondFeature	**In**: Face template data 2
cbSecondFeature	**In**: Length of the face template 2
score	**Out**: Returns the score (The range is 0-100. The higher the score, the greater the similarity)

Returns
Returns the Error Code. See the Appendix 1 for error code details.

Remarks
- Identification score range: 0-100. (Recommended threshold: 65)
- In actual functions, the threshold can be adjusted as required.
- Click here to view the Function List.

DestroyCompareHandle

Function Syntax

```
    int __stdcall DestroyCompareHandle(void *compareHandle);
```

Description
Releases the 1:1 face instance handle.

Parameter

Parameter	Description
compareHandle	**In**: Instance handle of face 1: 1 verification

Returns
Returns the Error Code. See the Appendix 1 for error code details.

Remarks
- If you no longer require using the 1: 1 face instance handle, you need to call this API to release it.
- Click here to view the Function List.

CreateSearchHandle

Function Syntax

```
int __stdcall CreateSearchHandle( void** searchHandle);
```

Description

Creates a 1:N high-speed cache (multiple high-speed caches can be created).

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

CreateGroup

Function Syntax

```
int __stdcall CreateGroup
    (
            void* searchHandle,
            unsigned int groupid
    );
```

Description

Creates a group in 1:N high-speed cache.

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache
groupid	**In**: Group ID

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

InsertFaceToGroup

Function Syntax

```
int __stdcall InsertFaceToGroup
   (
         void* searchHandle,
         unsigned int groupid,
         unsigned int faceID,
         const unsigned char * feature,
         int featureLen
   );
```

Description

Adds the face templates to a group specified by the 1:N high-speed cache.

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache
groupid	**In**: Group ID
faceID	**In:** Face ID
feature	**In**: Face template data
featureLen	**In**: Face template data length

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- This API is not a thread safe API.
- Click here to view the Function List.

DeleteFaceFromGroup

Function Syntax

```
int __stdcall DeleteFaceFromGroup
   (
         void* searchHandle,
         unsigned int groupid,
         unsigned int faceID
   );
```

Description

Deletes the face template in the group specified by the 1:N high-speed cache.

Parameter

Parameter	Description

searchHandle	**In**: Instance pointer to 1:N high-speed cache
groupid	**In**: Group ID
faceID	**In:** Face ID

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- This API is not a thread safe API.
- Click here to view the Function List.

Search

Function Syntax

```
int __stdcall Search
        (
                void* searchHandle,
                unsigned int groupid,
                const char * feature,
                int featureLen,
                PIdentifyFaceResult identifyFaceResult,
                int *maxRetCount
        );
```

Description

Performs 1:N recognition in the specified group ID of 1:N high-speed cache.

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache
groupid	**In**: Group ID
feature	**In:** Face template data for identification
featureLen	**In**: Face template data length
identifyFaceResult	**Out:** Returns identification result
maxRetCount	**In**: Maximum number of identification results returned
	Out: How many identification results actually returned

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Identification score range: 0-100. (Recommended threshold: 74)
- This API is not a thread safe API.

- In actual applications, the threshold can be adjusted as required.
- For related structures, see Structure Description
- Click here to view the Function List.

DestroyGroup

Function Syntax

```
int __stdcall DestroyGroup
    (
            void* searchHandle,
            unsigned int groupid
    );
```

Description

Deletes the specified group in 1:N high-speed cache.

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache
groupid	**In**: Group ID

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

DestroySearchHandle

Function Syntax

```
int __stdcall DestroySearchHandle(void* searchHandle);
```

Description

Releases 1:N high-speed cache resources.

Parameter

Parameter	Description
searchHandle	**In**: Instance pointer to 1:N high-speed cache

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- If you no longer require using the face 1: 1 instance handle, you need to call this API to release it
- Click here to view the Function List.

AnalyzeDetectResult

Function Syntax

int __stdcall AnalyzeDetectResult(void* detectResultHandle,unsigned char *faceInfo,unsigned int *cbFaceInfo);

Description

Analyzes the face information structure.

Parameter

Parameter	Description
detectResultHandle	**In**: Instance pointer of the face information structure (see DetectFaces interface)
faceInfo	**In**: Face information (returned data in json format), see Appendix 5 for specific json format
cbFaceInfo	**In/Out**: The length of the face information in: Pre-allocated memory size of faceInfo

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

AnalyzeFeatureResult

Function Syntax

int __stdcall AnalyzeFeatureResult(void* featureResult,unsigned int faceIndex,int featureType,unsigned char *featureInfo,unsigned int *cbFeatureInfo);

Description

Analyzes the face template structure.

Parameter

Parameter	Description
featureResult	**In**: Face template structure pointer (see ExtractFeature interface)
faceIndex	**In**: Face index (see DetectFaces. The value range is 0 to detectedFaces-1)
featureType	**In**: Face template type; 0 indicates the original template, 1indicates the

	integrated template (The parameter is generally passed to 0)
featureInfo	**Out**: Face template information (json format data), see Appendix 5 for specific json format
cbFeatureInfo	**In/Out**: The length of the face template information **In**: Pre-allocated memory size of the face template information Out: Actual length of the returned face template information

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After calling the ExtractFeature interface, call this interface.
- Click here to view the Function List.

AnalyzeLivenessResult

Function Syntax

```
int __stdcall AnalyzeLivenessResult(void* livenessResult,unsigned int faceIndex,float *livenessScore,int *reserved);
```

Description

Analyzes the face liveness feature structure.

Parameter

Parameter	Description
livenessResult	**In**: The pointer to the liveness feature structure of the face (see the PredictLiveness interface)
faceIndex	**In**: Face index (see DetectFaces. The value range is 0 to detectedFaces-1)
livenessScore	**Out**: The liveness detection value corresponding to the face index
reserved	**In**: Reserved parameters

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After calling the PredictLiveness interface, call this interface.
- Click here to view the Function List.

AnalyzeFaceAttributeResult

Function Syntax

```
int __stdcall AnalyzeFaceAttributeResult(void* attributeResult,unsigned int
```

faceIndex,unsigned char *attributeInfo,unsigned int *cbAttributeInfo);

Description

Analyzes the face attribute structure.

Parameter

Parameter	Description
attributeResult	**In**: Face attribute structure pointer (see PredictAttribute interface)
faceIndex	**In**: Face index (see DetectFaces. The value range is 0 to detectedFaces-1)
attributeInfo	**Out**: Face attribute information (json format data), see Appendix 5 for specific json format
cbAttributeInfo	**In/Out**: The length of the face attribute information **In**: The pre-allocated memory size of the face attribute information **Out:** The actual length of the face attribute information

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- After calling the PredictAttribute interface, call this interface.
- Click here to view the Function List.

ConverBioFeatureToStandFeature

Function Syntax

```
int __stdcall ConverBioFeatureToStandFeature(unsigned char *pStandFeature, int *pStandFeatureLen,unsigned char *pBioFeature, int bioFeatureLen);
```

Description

Converts the original template of the algorithm into integrated template.

Parameter

Parameter	Description
pStandFeature	**Out**: Integrated template data
pStandFeatureLen	**In:** Memory size allocated by the integrated template data **Out**: The length of the integrated template data
pBioFeature	**In:** Algorithm original template (see ExtractFeature or AnalyzeFeatureResult interface)
bioFeatureLen	**In/Out**: The original template length of the algorithm

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

ConverStandFeatureToBioFeature

Function Syntax

int __stdcall ConverStandFeatureToBioFeature(unsigned char *pStandFeature,unsigned char *pBioFeature, int *bioFeatureLen);

Description

Converts the integrated template to algorithm original template.

Parameter

Parameter	Description
pStandFeature	**In**: Integrated template data
pBioFeature	**Out:** Algorithm original template data
bioFeatureLen	**In:** The pre-allocated memory size of the algorithm original template data (pBioFeature) **Out**: The length of the algorithm original template data

Returns

Returns the Error Code. See the Appendix 1 for error code details.

Remarks

- Click here to view the Function List.

6 Appendix

6.1 Appendix 1: Error Code

Error Code	Description
0	Call succeeded
1	Call failed
2	Runs out of chip trial period
3	Chip verification failed
4	Parameter error
5	Model path is empty
6	Memory allocation exception, such as malloc failure.
10	Cache DB is full
11	Please try again
40	Unsupported API
41	Obsolete API
50	Model file error
51	Unsupported model type
52	Unsupported model version
53	Lack of necessary model
54	Model handle error
55	Too many models loaded
1101	The algorithm does not detect the face
1102	An unknown error occurred in the algorithm
1103	Low image quality
1104	Error in extracting face feature data
1105	Unqualified face in the image
2000	Uninitialized algorithm
2001	No face detected
2002	Insufficient memory allocated
3101	Internal error
3102	Unknown error
3103	Null pointer
3104	Unsupported configuration
3105	Unsupported model
3106	File is damaged
3107	Out of parameter range
3108	Invalid setting
3109	File not found
3110	Invalid parameter
3111	Invalid type
3112	Unsupported operation
3113	Invalid license
3114	Invalid setting

3115	Unknown global option error
3116	Unauthorized
3117	Duplicate face ID
3118	Invalid face ID
3119	Face ID not found
3120	Error handle

6.2 Appendix 2: Glossary

This glossary will help you understand the basic functions of visible light facial recognition applications and quickly complete the integrated development of visible light facial recognition applications.

Identification/Verification template

Used for 1:1 verification or 1:N identification recognition, that is, the face template obtained by calling ExtractFeature API.

Registration template

To be added to the low-level library (high-speed cache), that is, the face template obtained by calling ExtractFeature API.

Face registration

The process of collecting face images of users via the face module/collector device. The collected images are then processed to extract a face template. As a result, the template transfer to the background system in database, which can be used for later identification.

(1:1) Face verification

1:1 face verification, also known as face verification, is a process of confirming the identity of a user by comparing their user ID and face template. This process is used to determine whether a given set of verification templates are extracted from the same face as the registered template.

(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.

6.3 Appendix 3: License Application and Activation

License Application

In the SDK development kit, there is a license folder.

Open the CMD console and navigate to the current license folder path, and then run the following command;

```
CMD> hasp_update_expire_xp32.exe f fingerprint.c2v
```

If the execution is successful a fingerprint.c2v file will be generated under the current path.

Send this file to technical support to apply for a license file.

License Activation

After receiving fingerprint.v2c license file, put it in the license folder path of the SDK development kit.

Open the CMD console and navigate to the current license folder path, and then run the following command;

```
CMD> hasp_update_expire_xp32.exe u fingerprint.v2c
```

The console prints: ***`HelloWorld LIuK`***

It means activation is successful, otherwise the activation is failed.

6.4 Appendix 4: Back up Registration Image

Back up Registration Image

It is recommended that users must save the registration images when registering faces, and it may be required to re-extract features when the algorithm model is upgraded.

6.5 Appendix 5: Json Format Description

Face template information Json format:

```
{
"featureinfo":
 {
 "major": 58, // Template major version number
 "minor": 10, // Template minor version number
 featuredatabase64:"ADF...", // Face template Base64 format data
 featuredatabase64len:2752 // The length of the face template Base64 format
data
 }
}
```

Face information Json format:

```
{
 "facecount": 5,// Number of faces
 // Face information array
 "faceinfo":
 [
 {
 "left": 1, // The X coordinate of the upper left corner of the face
rectangle
 "top": 2, // The Y coordinate of the upper left corner of the face
rectangle
 "right": 1, // The X coordinate of the lower right corner of the face
rectangle
 "bottom": 2, // The Y coordinate of the lower right corner of the face
rectangle
 "extentleft": 0, // Reserved parameters
 "extenttop": 0, // Reserved parameters
 "extentright": 0, // Reserved parameters
 "extentbottom":0, // Reserved parameters
 "roll": 2.0, //Roll angle of the face
 "pitch": 2.0, //Pitch angle of the face
 "yaw": 1.0,//Yaw angle of the face
 "blur": 2.4, // Face blur degree (value range: 0~1.0, 0 means the
clearest, 1 means the most blurry)
 "brightness": 3, // Face brightness
 "brightnessdeviation": 4 // Standard deviation of face brightness
 "face_completeness":0.90 // Face completeness, range (0~1), the higher the score, the better the face completeness
 },
 {
 "left": 1,
 "top": 2,
 "right": 1,
 "bottom": 2,
 "extentleft": 1,
 "extenttop": 2,
 "extentright": 1,
 "extentbottom": 2,
 "roll": 2.0,
 "pitch": 2.0,
 "yaw": 1.0,
 "blur": 2.4,
 "brightness": 3,
 "brightnessdeviation": 4,
 "face_completeness":0.90
 }
 ]
}
```

Face attribute Json format:

```
{
"faceattributeinfo":
 {
 "gender": 1,// 0: means unknown, 1: means male, 2: means female
 "age": 10,// Age
 "maskstatus":0// 0: means unknown, 1: means not wearing a mask, 2: means wearing a mask
 }
}
```

190 Bluegrass Valley Pkwy,

Alpharetta, Georgia 30005. USA

E-mail: info@armatura.us

www.armatura.us