ARMATURA

API Development Manual:

AMTFacePro SDK for Android

API Version: 5.62

Doc Version: 1.0

October 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.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 **AMTFacePro SDK For Android**.

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

Document Conventions

Conventions used in this manual are listed below:

GUI Conventions

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

Symbols

Convention	Description
	This 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 AMTFacePro Algorithm Description

Facial recognition is a system that automatically recognizes a human face from an image or video. As one of the earliest biometric technologies, facial recognition has many advantages over other biometric technologies: innate, non-invasive, and easy to use. Facial recognition has become increasingly relevant today due to the rapid growth of industrial technology (such as digital video cameras, mobile network equipment) and growing security requirements, and widely used in various systems, including attendance, security, video monitoring, and so on.

As a visible light source-based face recognition algorithm, AMTFacePro is a fast and accurate 1:1 and 1:N algorithm. It is fully open to software developers and system integrators. It is possible to configure different SDK versions according to markets and customer needs. Besides, different SDK versions support consistent comparison and recognition of face templates in various platforms.

1.1 AMTFacePro SDK Technical Description

Image

- To obtain a high-quality face template and speed, set the detection distance according to the actual application scenario.
- The minimum imaging resolution recommended for face registration combined with recognition is 640 x 480 pixels. The SDK supports resolutions that are smaller than the specified value, but this will decrease the accuracy of face registration and recognition, which will affect the consistency of the face template.
- Use multiple images during the registration process, which in this way the quality of the face template improves and enhances the quality and reliability of face recognition.

Illumination

- Please consider the controllable and uncontrollable lighting conditions. Pay attention to the following typical conditions:

  The front direct light and the diffuse light have the same light distribution for each angle of the face and the shadow of the entire face area.

  Certain types of lighting can also cause reflections on glasses or face.

Facial posture

Facial recognition algorithms support multiple poses.

- ±30 degrees of head tilt, ±25 degrees are standard values, appropriate for most front-closing facial images.
- ±30 degrees of pitch and front position deviation, ±25 degrees is a standard value. Increase the allowable tolerance for the head to ±30 degrees if there are several low-angle images of the same face in the face registration process.
- Shake your head and offset from the front position by ±30 degrees.
- A deviation of ±15 degrees is a standard value, and this deviation is sufficient for a face image that is close to the front. Recommended registering multiple photos of the face in the database to support a deviation angle of ±30 degrees from the front of the face.

Facial expression

The facial algorithm maximizes the accuracy of face recognition under certain unnatural facial expressions. Examples of unusual expressions are as follows (allowed but not recommended):

- Laughing (exposed part of the tooth or mouth)
- Eyebrows rise
- Closed eyes
- Frowning eyebrows

Glasses, makeup, hair, beard, and mustache

To ensure the quality of face recognition, the algorithm SDK supports the status that part of the face covered by glasses or hair:

- **Glasses**: Ordinary rimmed glasses cover a part of the face causing some facial features invisible, reducing the quality of face recognition.
- **Contact lens**: Contact lenses do not affect face recognition. However, people wearing contact lenses may also wear ordinary lens glasses.
- Heavy makeup may hide or distort facial features.
- **Hairstyle**: Some hairstyles may cover part of the face.
- Changes in facial hairstyle may require additional face registration, especially when the beard/ mustache grows out or shaved.

1.2 AMTFacePro SDK Main Function

Face Detection

The Face Algorithm SDK provides fast, high-accuracy face detection. It is generally applicable to pictures and live video streams and can detect faces of not less than 36*36 pixels.

Face key point detection

Face Algorithm SDK face key point detection can accurately locate key areas of the face, including eyes, sharp chin, facial contours, so on and supports a certain degree of face occlusion.

Face attribute analysis

Analyze the gender, age, and whether the target face is wearing a mask.

Live face detection

Provides a single-frame photo live detection interface and can also perform motion-coordinated live detection based on face pose estimation.

Face recognition

1:1 Face Verification: Face algorithm SDK of face verification technology is used for application scenarios such as login verification and identity recognition. Help users quickly decide whether two images match, determine whether the target face is the face detected in the video, support authentication of real-time recognition, and incorporates identity and face binding features.

1:N Face Identification: Face algorithm SDK of face identification technology can automatically identify the face identity in photos and video streams, and its recognition speed and accuracy are among the world's outstanding levels.

2 AMTFacePro SDK Architecture and Installation

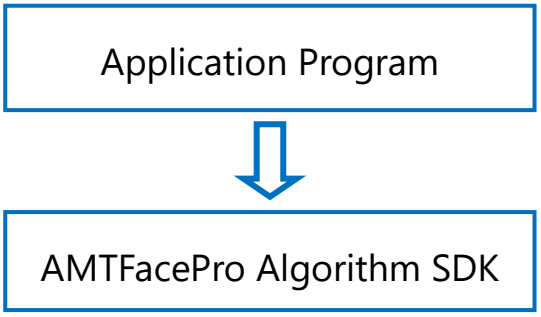
2.1 AMTFacePro SDK Architecture

The AMTFacePro SDK Android version mainly exists in the form of a java interface. The user can develop an application based on visible light face recognition using the Android application development language (java).

Files Included

Operating System	Files	Description
Android	All files under libs	AMTFacePro Algorithm library

SDK Architecture



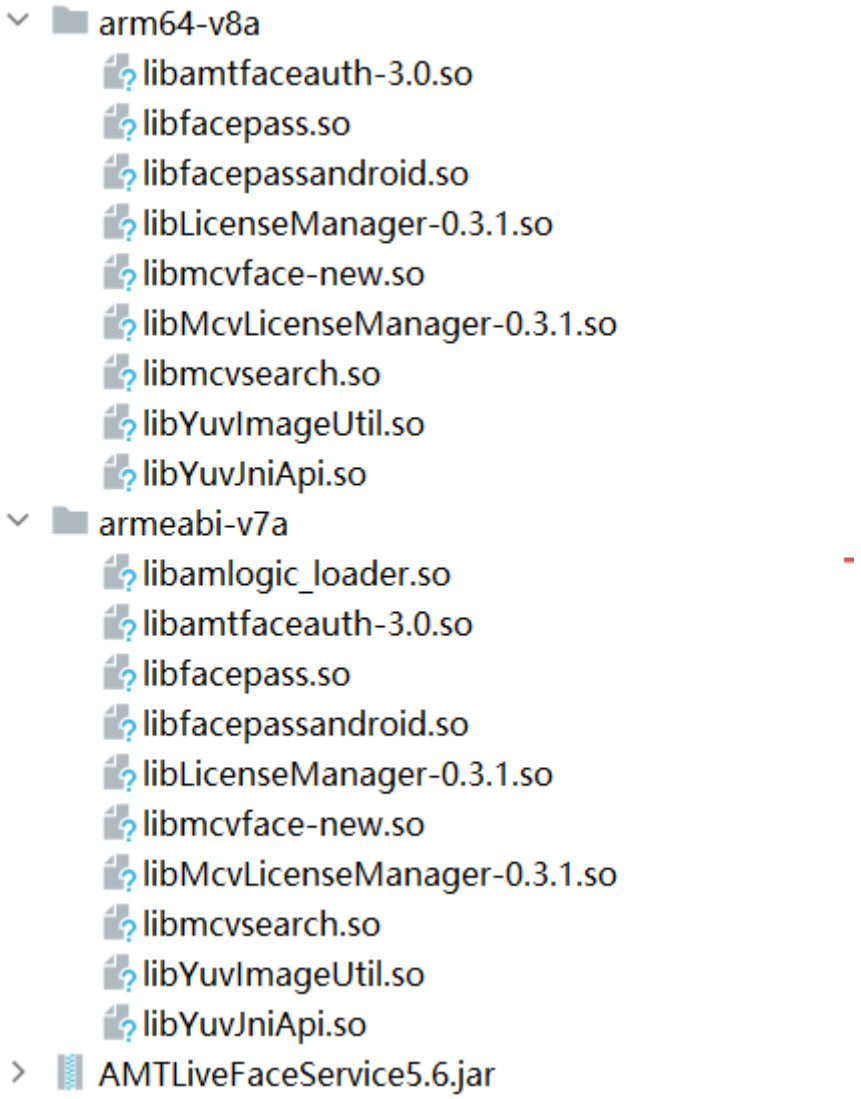
2.2 Software Installation

Before installing the AMTFacePro SDK, make sure your operating system or mobile device meets the software operations requirements.

Package the AMTFacePro algorithm library into the application, and users may have different packaging methods using various development tools. The following is an example to describe the use of AMTFacePro SDK in the Android Studio IDE development environment.

- Copy the ***.so** and ***.jar** files to the libs directory of the Android project. The ***.so** library files get

saved in different directories according to the CPU architecture.



2.3 License Application and Usage Matters

This SDK uses device hardware information or encryption chip binding. The two licensing methods have explained separately below.

2.3.1 Device Hardware Information Binding

Due to the need for reading device hardware information and read and write permissions, please configure at least the following permissions in the list:

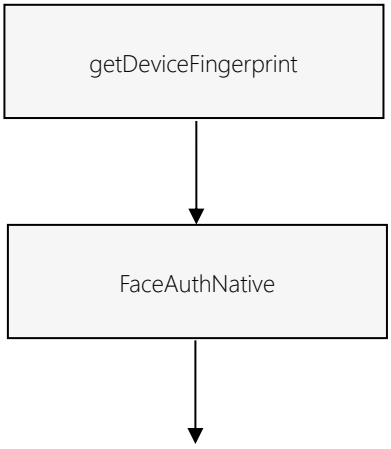
```
<uses-permission android:name="android.permission.CAMERA" />
```

```
<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" /> Please add this permission for Android 10 and above

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Please add the following code for Android 10 and above and grant full Sdcard access.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {

        if (!Environment.isExternalStorageManager()) {

            startActivity(Intent(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION))

        }

    }
```

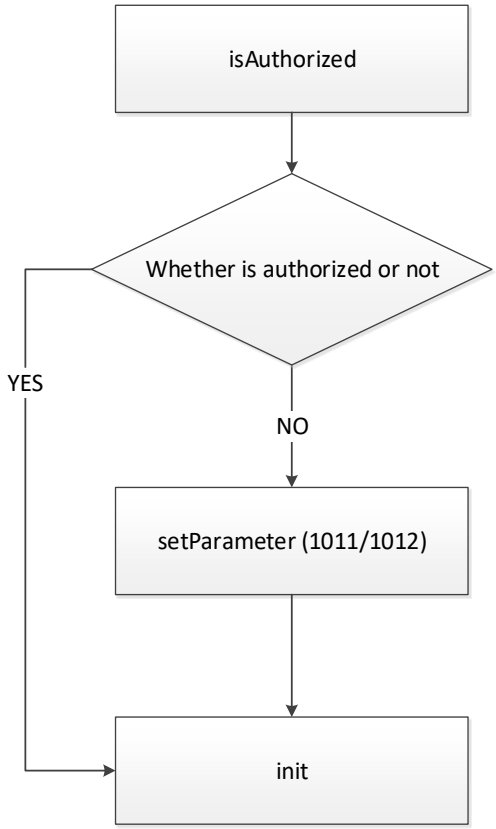
License Application Process



Process Description

- Call **getDeviceFingerprint** to get device hardware information.
- Apply for a license with the obtained device hardware information.
- Obtain license file.

License Activation Process



Process Description

- Call **isAuthorized** to determine whether the License Activation has an authorized state (can directly call **init** to initialize when it has the authorized state).
- Call **setParameter** when not authorized to set the license file name or license data to SDK through 1011/1012 (**Appendix 2 – Parameter Code**) parameter.
- Call **init** to initialize the algorithm (algorithm initialization completes the final activation process)

3 AMTFacePro SDK Interface Class Details

3.1 Function List

Function Name	Description
version	Gets the version number
getLastError	Returns the latest error message
isAuthorized	Checks whether the current device is authorized
getHardwareId	Gets the machine code
getDeviceFingerprint	Gets the device hardware information
setParameter	Sets the parameters
getParameter	Obtains the parameters
init	Initializes algorithm library
terminate	Releases the algorithm resources
detectFacesFromNV21	Detects the face in NV21 format
detectFacesFromBitmap	Detects the face.
getFaceContext	Gets the specified face instance pointer
getLiveness	Gets the face liveliness score
getFacePose	Gets the face pose (in 3 angles).
getFaceICaoFeature	Gets the ICAO features
getFaceRect	Gets a rectangle that detects faces
extractTemplate	Extracts the facial template
closeFaceContext	Releases the face instance objects
verify	Process the 1:1 Face comparison
dbAdd	Adds the face template to 1:N cache.
dbDel	Adds the face template to the default 1:N buffer
dbClear	Clears the default 1:N cache
dbCount	Gets the default number of 1:N cache template.
dbIdentify	Gets the default 1:N high speed buffer to perform 1:N recognition

version

Function Syntax

```
public static int version
        (
            byte[] version,
            int[] size
    );
```

Description

This function gets the version number.

Parameters

Parameter	Description
version	**Out**: Returns version number (it is recommended to pre allocate more than 128 bytes).
size	**In**: version memory size (bytes).
	Out: Actual version length returned.

Returns

Error Code	Refer to - Appendix 1: Error Code

Example

```
byte[] version = new byte[256];
int[] size = new int[1];
size[0] = 256;
if (0 == AMTLiveFaceService.version(version, size))
        {
            String verStr = new String(version, 0, size[0]);
    }
```

Remarks

- Click here to view the Function List.

getLastError

Function Syntax

```
public static int getLastError
        (
            long context,
            byte[] lasterror,
            int[] size
        );
```

Description

This function returns the latest error message.

Parameters

Parameter	Description
context	**In**: An algorithm instance pointer (allowing NULL to be passed) is the last error of the instance when it is not NULL (and the interface is called when the error code is 11 to get the error description)
lasterror	**Out:** Error message (recommended pre-allocation of 256 bytes, enough use)
size	**In**: version memory size (bytes)
	Out: Actual last error length returned.

Returns

Error Code	Refer to - Appendix 1: Error Code

- If the interface returns failure, the general error is that the allocated memory is insufficient.

Example

```
byte[] lasterror = new byte[256];
int[] size = new int[1];
size[0] = 256;
if (0 == AMTLiveFaceService.getHardwareId(context, lasterror, size))
        {
    String errStr = new String(lasterror, 0, size[0]);
```

```
}
```

Remarks

- When the interface used does not need to pass a context algorithm instance pointer, the context parameter can pass 0 when the interface is called, such as interface: init, etc. When these interfaces call getLastError, context passes 0.
- Click here to view the Function List.

isAuthorized

Function Syntax

```
public static boolean isAuthorized();
```

Description

This function checks whether the current device is authorized.

Returns

true	Authorized
false	Unauthorized

Remarks

- Devices with legal encryption chips always returns true (authorized status).
- Click here to view the Function List.

getHardwareId

Function Syntax

```
public static int getHardwareId
        (
            byte[] hwid,
```

```
        int[] size
    );
```

Description

This function gets the machine code.

Parameters

Parameter	Description
hwid	**Out**: Returns the machine code (recommended to allocate 256 bytes)
size	**In**: hwid Memory size (bytes)
	Out: Actual hwid length returned.

Returns

Error Code	Refer to - Appendix 1: Error Code

Example

```
byte[] hwid = new byte[256];
int[] size = new int[1];
size[0] = 256;
if (0 == AMTLiveFaceService.getHardwareId(hwid, size))
        {
            String hwidStr = new String(hwid, 0, size[0]);
    }
```

Remarks

- The machine code has nothing to do with the actual binding hardware information. It is only used here to assist the user in associating the machine and the license file.
- Click here to view the Function List.

getDeviceFingerprint

Function Syntax

```
public static int getDeviceFingerprint
        (
            byte[] devFp,
            int[] size
        );
```

Description

This function gets the device hardware information.

Parameters

Parameter	Description
devFp	**Out**: Returns device hardware information (recommended pre-allocation of 32*1024 bytes)
size	**In**: devFp Memory size (bytes)
	Out: Actual devFp length returned.

Returns

Error Code	Refer to - Appendix 1: Error Code

Example

```
byte[] devFp = new byte[32*1024];
int[] size = new int[1];
size[0] = 32*1024;
if (0 == AMTLiveFaceService.getDeviceFingerprint(devFp, size))
    {
        String devFpStr = new String(devFp, 0, size[0]);
}
```

Remarks

- Save **devFp** as a file or other form and send it to the business to apply for license.
- Click here to view the Function List.

setParameter

Function Syntax

```
public static int setParameter
        (
            long context,
            int code,
            byte[] value,
            int size
        );
```

Description

This function sets the parameters.

Parameters

Parameter	Description
context	**In**: Algorithm instance pointer
code	**In:** Parameter code (see Appendix 2)
value	**In**: Parameter value
size	**In:** Data length (bytes)

Returns

Error Code	Refer to - Appendix 1: Error Code

Remarks

- Set the maximum number of faces detected and the 1:1 comparison threshold.
- Here the parameter value is a pure numeric string. For example, the parameter value should be set to: "68".
- Click here to view the Function List.

getParameter

Function Syntax

```
public static int getParameter
```

```
(
    long context,
    int code,
    byte[] value,
    int[] size
);
```

Description

This function obtains the parameters.

Parameters

Parameter	Description
context	**In**: Algorithm instance pointer
code	**In**: Parameter code (see Appendix 2)
value	**Out:** Parameter value
size	**In:** Allocated data length of the value
	Out: Returned actual parameter data length

Returns

Error Code	Refer to - Appendix 1: Error Code

Remarks

- Get the 1:1 comparison threshold, and the parameter value is a pure numeric string. For example, to get the 1:1 threshold, the return parameter value is "76".
- Click here to view the Function List.

init

Function Syntax

public static int init(long[] context);

Description

This function initializes algorithm library.

Parameters

Parameter	Description
context	**Out**: Returns algorithm instance pointer (context[0])

Returns

Error Code	Refer to - Appendix 1: Error Code

Example

```
long context[] = new long[1];
int ret = AMTLiveFaceService.init(context);
    if (0 == ret)
            {
                System.out.print("Init succ, context=" + context[0]);
        }
    else
            {
                System.out.print("Init failed, error code=" + ret);
        }
```

Remarks

- After the initial interface gets invoked successfully, invoke the **setParameter** to set the parameters related to face detection and recognition. The specific setting method and the related parameters can be described by referring to the setParameter interface.
- Click here to view the Function List.

terminate

Function

public static int terminate (long context);

Description

This function releases the algorithm resources

Parameter

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

Returns

Error code	Refer to: Error Code

Remarks

- Click here to view the Function List.

detectFacesFromNV21

Function

```
public static int detectFacesFromNV21
        (
            long context,
            byte[] rawImage,
            int width,
            int height,
            int[] detectedFaces
        );
```

Description

This function detects the face in NV21 format.

Parameter

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

rawImage	**In**: NV21 image data
width	**In:** Image width
height	**In:** Image height
detectedFaces	**Out**: The number of faces detected (is < = maximum number of faces detected). The default maximum face detection number is 1.

Returns

Error code	Refer to: Error Code

Remarks

- Since the default output of video stream is in NV21 format, this interface is added for easy use.
- Click here to view the Function List.

detectFacesFromBitmap

Function

```
public static int detectFacesFromBitmap
        (
            long context,
            Bitmap bitmap,
            int[] detectedFaces
    );
```

Description

This function detects the face.

Parameter

Parameter	Description
context	**In:** Algorithm instance pointer
bitmap	**In**: image
detectedFaces	**Out**: The number of faces detected (is < = the maximum number of faces detected. The default maximum face detection number is 1.

Returns

Error code	Refer to: Error Code

Remarks

- The second-generation ID card photos can be detected through Bitmap objects.
- Click here to view the Function List.

getFaceContext

Function

```
public static int getFaceContext
        (
            long context,
            int faceIdx,
            long[] faceContext
    );
```

Description

This function gets the specified face instance pointer.

Parameter

Parameter	Description
context	**In:** Algorithm instance pointer.
faceIdx	**In**: Face index(see detectFacesFromNV21, 0~[detectedFaces-1]).
faceContext	**Out**: Returns face instance pointer.

Returns

Error code	Refer to: Error Code

Remarks

- Click here to view the Function List.

getLiveness

Function

```
public static int getLiveness
        (
            long faceContext,
            int[] score
        );
```

Description

This function gets the face liveliness score.

Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.
score	**Out:** Liveness detection fraction

Returns

Error code	Refer to: Error Code

Remarks

- The recommended score of liveliness detection is 65; and < 65 is recognized as false face.
- Click here to view the Function List.

getFacePose

Function

```
public static int getFacePose
        (
            long faceContext,
            float[] yaw,
```

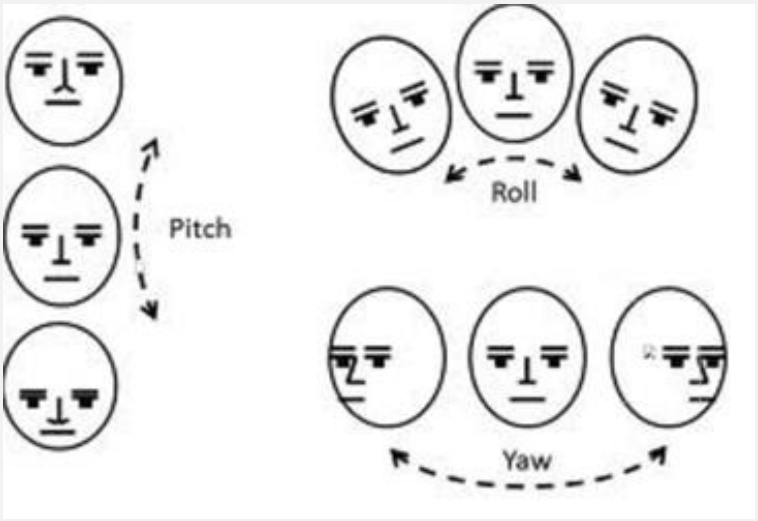
```
        float[] pitch,
        float[] roll
    );
```

Description

This function gets the face pose (in 3 angles).

Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.
yaw	**Out:** The rotation angle centered on the nose between [-90, + 90]. Where 0 means positive face, clockwise is positive, counterclockwise is negative, the algorithm cannot detect the face when the angle is too large, usually need to be controlled within 45°.
pitch	**Out:** Up and down pitch angle value, the value is between [-90, +90]. Where 0 means positive face, head up is positive, head down is negative, the algorithm cannot detect the face when the angle is too large, usually need to be controlled within 45°
roll	**Out:** Left and right roll angle value, between [-90, +90]. Where 0 means positive face, left turn is positive, right turn is negative, the algorithm cannot detect the face when the angle is too large, usually need to be controlled within 45°.



Returns

Error code	Refer to: Error Code

Remarks

- Click here to view the Function List.

getFaceICaoFeature

Function

```
public static int getFaceICaoFeature
        (
            long faceContext,
            int featureID,
            int[] score
        );
```

Description

This function gets the ICAO features.

Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.
featureID	**In:** Feature ID
score	**Out:** Returns score

Returns

Error code	Refer to: Error Code

Remarks

- For AMTFACE_ICaoFEATUREID, see Appendix 3
- Click here to view the Function List.

getFaceRect

Function

```
public static int getFaceRect
        (
            long faceContext,
            int[] points,
            int cntPx
        );
```

Description

Gets a rectangle that detects faces.

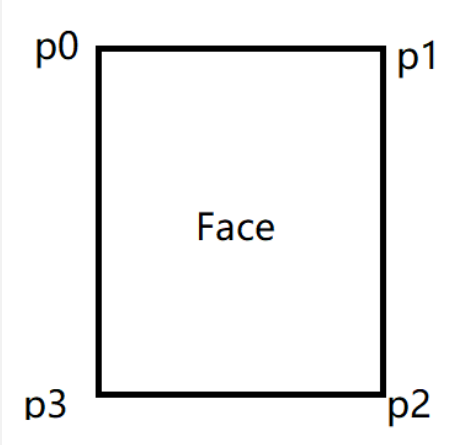
Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.
points	**Out:** Four coordinate points of rectangle box p0.x p0.y p1.x p1.y p2.x p2.y p3.x p3.y Sequential arrangement (clockwise)
cntPx	**Out:** points Array size(8)

Returns

Error code	Refer to: Error Code

Example

Coordinate point example


Remarks

- Picture zoom and rotate, convert the coordinates according to the actual situation
- Click here to view the Function List.

extractTemplate

Function

```
public static int extractTemplate
        (
            long faceContext,
            byte[] template,
            int[] size,
            int[] resverd
        );
```

Description

This function extracts the facial template.

Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.
template	**Out:** Face template (recommended to allocate at least 256 bytes)
size	**In:** Template Memory allocation size
	Out: Returns actual template data length
resverd	**Out**: This parameter is a reserved parameter

Returns

Error code	Refer to: Error Code

Example

```
int ret = 0;
byte[] template = new byte[256];
int[] size = new int[1];
int[] resverd = new int[1];
size[0] = 256;
ret = AMTLiveFaceService.extractTemplate(faceContext, template, size, resverd);
```

Remarks

- Click here to view the Function List.

closeFaceContext

Function

public static int closeFaceContext(long faceContext);

Description

This function releases the face instance objects.

Parameter

Parameter	Description
faceContext	**In**: Face instance pointer.

Returns

Error code	Refer to: Error Code

Remarks

- Click here to view the Function List.

verify

Function

```
public static int verify
        (
            long context,
            byte[] regTemplate,
            byte[] verTemplate,
            int[] score
        );
```

Description

This function process the 1:1 Face comparison.

Parameter

Parameter	Description
context	**In**: Algorithm instance pointer.
regTemplate	**In**: Registration template
verTemplate	**In:** Verification template
score	**Out**: Returns score

Returns

Error code	Refer to: Error Code

Example

```
int ret = 0;
int[] score = new int[1];
ret = AMTLiveFaceService.verify(context, regTemplate, verTemplate, score);
```

Remarks

- The 1:1 comparison threshold is 65. And if exceeds the defined value, then the comparison is successful.
- Comparison score range: 0~100, see Appendix 4 for details
- Click here to view the Function List.

dbAdd

Function

```
public static int dbAdd
        (
            long context,
            String faceID,
            byte[] regTemplate
        );
```

Description

This function adds the face template to 1:N cache.

Parameter

Parameter	Description
context	**In**: Algorithm instance pointer.
faceID	**In**: Face ID
regTemplate	**In:** Registration template

Returns

Error code	Refer to: Error Code

Remarks

- Non-thread safe interface (note the memory db read and write is in protection)
- Face capacity, 50,000 face ID is recommended
- Click here to view the Function List.

dbDel

Function

```
public static int dbDel
        (
            long context,
            String faceID
    );
```

Description

This function adds the face template to the default 1:N buffer.

Parameter

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

Returns

Error code	Refer to: Error Code

Remarks

- Non thread safe interface (note the memory db read and write is in protection)
- Click here to view the Function List.

dbClear

Function

public static int dbClear(long context);

Description

This function clears the default 1:N cache.

Parameter

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

Returns

Error code	Refer to: Error Code

Remarks

- Non thread safe interface (note the memory db read and write is in protection)
- Click here to view the Function List.

dbCount

Function

```
public static int dbCount
        (
            long context,
            int[] count
        );
```

Description

This function gets the default number of 1:N cache template.

Parameter

Parameter	Description
context	**In**: Algorithm instance pointer.
count	**Out:** Number of templates returned

Returns

Error code	Refer to: Error Code

Remarks

- Non thread safe interface (note the memory db read and write is in protection)
- Click here to view the Function List.

dbIdentify

Function

```
public static int dbIdentify
        (
            long context,
            byte[] verTemplate,
            byte[] faceID,
```

```
                int[] score,
                Int[] identifyScore,
                int[] maxRetCount,
                int minScore,
                int maxScore
        );
```

Description

This function takes the default 1:N high speed buffer to perform 1:N recognition

Parameter

Parameter	Description
context	**In**: Algorithm instance pointer.
verTemplate	**In:** Verification template
faceID	**Out**: Returns the face ID
score	**Out:** Returns comparison score
identifyScore	**Out:** Returns liveness score
maxRetCount	**In:** Maximum number of returns
	Out: Actual number of returns
minScore	**In**: Minimum matching score. The recognition will be successful only when the similarity between the recognized face and a face template in the database meets the minimum matching score value.
maxScore	**In**: The recognition success returns immediately only when the similarity between the recognized face and a face template in the database reaches this maximum score value.

Returns

Error code	Refer to: Error Code

Example

```
int ret = 0;
int[] score = new int[1];
byte[] faceIDS = new byte[256];
int[] maxRetCount = new int[1];
maxRetCount[0] = 1; //only returns 1 face ID
ret = AMTLiveFaceService.dbIdentify(context, verTemplate, faceIDS, score, maxRetCount, 70, 100);
```

Remarks

- Non thread safe interface (note memory db read and write protection)
- Comparison score range: 0~100, see Appendix 4 for details.
- minScore and maxScore Parameter description: The algorithm compares all the templates loaded into the memory database in a cycle. In the process of cycle comparison, when the similarity between the recognized face and a face template in the memory database reaches the maxScore value, the recognition is successful and exits the cycle comparison immediately. When the similarity between the recognized face and a face template in the memory database meets the minScore value, the current face ID number gets saved. This cyclic comparison will get continued until all the templates get compared. And the face IDs set by Parameter maxRetCount are returned according to the similarity from high to low.
- Click here to view the Function List.

4 Workflow Description

4.1 Algorithm Authorization and Process

4.1.1 Authorization Process

- For Authorization Process detail, refer to License Application and Usage Matters.

4.1.2 Restore to Factory Settings / Reburn the System

For devices authorized by device hardware information, remember to restore factory settings and make sure not to select the "format SD card" option to avoid license loss. And if the license is lost, you need to apply again.

Similarly, if the license file gets lost due to operations such as "Reburn" the system, it is also necessary to reapply for authorization.

The device hardware information authorization needs to get obtained again when applying for re-authorization.

4.2 Algorithm Initialization

4.2.1 Initialization Interface Description

Initialize the face recognition engine, if successful, returns zero. Call this function before calling any other functions. For more details on the interface description, please refer to the interface description corresponding to Init Function.

```
public static int init(long[] context);
```

4.2.2 Initialization Example Program Description

Example

```
long context[] = new long[1];
int ret = AMTLiveFaceService.init(context);
if (0 == ret)
        {
                System.out.print("Init succ, context=" + context[0]);
}
else
{
        System.out.print("isAuthorized=" + AMTLiveFaceService.isAuthorized());
        System.out.print("Init failed, error code=" + ret);
}
```

4.3 Face Detection

Detects the number of faces through Bitmap/NV21 data and returns zero if successful. Then checks whether detectedFaces[0] is greater than 1.

- Call the Face detection interface.

- For more details, refer to **detectFacesFromNV21**/**detectFacesFromBitmap** function description.
- Obtain the face instance
- For more details, refer to **getFaceContext**.

4.4 Extract Template

- Call **Face Detection** to detect the face and get the pointer of the face instance.
- Call **extractTemplate** to extract facial features.

4.5 Register Face

4.5.1 Face Registration Interface Description

After successfully detecting the number of faces, the instance pointer of a single face can be obtained, and the face template can be extracted according to the face instance pointer. And after successful extraction, the user can proceed to register.

//Get a single face instance pointer interface. For detailed interface description, please refer to the corresponding interface description of **getFaceContext**.

```
public static int getFaceContext
                (
                        long context,
                        int faceIdx,
                        long[] faceContext
        );
```

//Extract the face template. For detailed interface description, please refer to the corresponding interface description of **extractTemplate**.

```
public static int extractTemplate
                (
                        long faceContext,
```

```
                byte[] template,
                int[] size,
                int[] resverd
        );
```

If 1:N comparison process is required, the obtained face template needs to be registered in the cache. For detailed interface description, please refer to the interface description corresponding function **dbAdd**.

```
public static int dbAdd
                (
                        long context,
                        String faceID,
                        byte[] regTemplate
        );
```

4.5.2 Face Registration Example Program Description

1. After detecting the number of faces, the face pointer of a single face is obtained, and the face template is extracted.

Example

//Get a single face instance pointer (the instance face index is: 0), instanceContext is the instance pointer to the successful initialization of algorithm.

//Range of face index: detectFaces, 0~detectedFaces-1

```
long[] faceContext = new long[1];

retCode = AMTLiveFaceService.getFaceContext(instanceContext, 0, faceContext);

if(0 == retCode )

{

//Extracts the face template (face index is: 0) and suggested to pre-allocate 256 bytes to store the face template.

byte[] template = new byte[256];
```

```
size = new int[1];

size[0] = 256;

int[] resverd = new int[1];
// faceContext [0] is the pointer of the face instance obtained successfully at first, after which the face template can get registered.

retCode = AMTLiveFaceService.extractTemplate(faceContext[0], template, size, resverd);
}
```

2. If 1:N comparison is required, the obtained face template is required to register in the 1:N cache memory.

   Example

   // instanceContext is the instance pointer after initialization of the algorithm. Template is the face template to be added to the 1:N cache. "**Reg1**" is the user ID to register to the 1:N cache.

   ```
   retCode = AMTLiveFaceService.dbAdd(instanceContext, "Reg1", template);
   ```

3. Backup registration photos

   Customers are strongly advised to save the registration photos when they register their faces, and the features need to be extracted again when the algorithm model gets upgraded.

4.6 Face Comparison

4.6.1 Face Comparison Program Description

1:1 Face verification

//For interface details, see the corresponding interface description in the function **verify**.

```
public static int verify
                    (
                        long context,
                        byte[] regTemplate,
                        byte[] verTemplate,
                        int[] score
                )
```

1:N Face identification

//For interface details, see the corresponding interface description in **dbIdentify.**

```
public static int dbIdentify
                    (
                        long context,
                        byte[] verTemplate,
                        byte[] faceIDs,
                        int[] score,
                        int[] maxRetCount,
                        int minScore,
                        int maxScore
                )
```

4.6.2 Face Comparison Example Program Description

1:1 Face verification

```
int score = 0;
// instanceContext is the instance pointer after initialization of the algorithm. regTemplate, verTemplate are the face templates to be compared.
int ret = 0;
//Stores the returned comparison score.
int[] score = new int[1];
ret = AMTLiveFaceService.verify(instanceContext, regTemplate, verTemplate, score);
```

1:N Face identification (returns single face ID)

```
int ret = 0;
int[] score = new int[1]; // Stores the returned comparison score.
byte[] faceIDS = new byte[256]; //Stores the returned face ID.
int[] maxRetCount = new int[1];
maxRetCount[0] = 1; //Only returns 1 face ID
// instanceContext is the instance pointer after initialization of the algorithm. verTemplate is the face template to be compared.
ret = AMTLiveFaceService.dbIdentify(instanceContext, verTemplate, faceIDS, score, maxRetCount, 70, 100);
```

Remarks

minScore and **maxScore Parameter Description**:

- The algorithm compares all the templates loaded into the memory database in a cycle.
- In the process of cycle comparison, when the similarity between the recognized face and a face template in the memory database reaches the maxScore value, the recognition is successful and exits the cycle comparison immediately.
- When the similarity between the recognized face and a face template in the memory database meets the minScore value, the current face ID number gets saved.
- This cyclic comparison will get continued until all the templates get compared.
- And the face IDs set by Parameter maxRetCount are returned according to the similarity from high to low.

1:N face identification (returns multiple face IDs)

```
int ret = 0;

int[] maxRetCount = new int[1];

maxRetCount[0] = 6; //Returns 6 faces IDs

int[] score = new int[maxRetCount[0]]; //Store the corresponding score of the returned face ID.

byte[] faceIDS = new byte[4096];//Face ID multiple records are separated by \t.
// instanceContext is the instance pointer after initialization of the algorithm. verTemplate is the face template
to be compared.
ret = AMTLiveFaceService.dbIdentify(context, verTemplate, faceIDS, score, maxRetCount, 70, 100);
```

Remarks

minScore and **maxScore Parameter Description**:

- The algorithm compares all the templates loaded into the memory database in a cycle.
- In the process of cycle comparison, when the similarity between the recognized face and a face template in the memory database reaches the maxScore value, the recognition is successful and exits the cycle comparison immediately.
- When the similarity between the recognized face and a face template in the memory database meets the minScore value, the current face ID number gets saved.
- This cyclic comparison will get continued until all the templates get compared.
- And the face IDs set by Parameter maxRetCount are returned according to the similarity from high to low.

4.7 Liveliness Detection

After successfully detecting the number of faces, the instance pointer of a single face is obtained, and the liveliness detection interface is invoked to judge whether the face is false or not.

Example

```
//Gets a single face instance pointer (the instance face index is: 0). The instanceContext is the instance pointer after the algorithm is initialized successfully
//Range of the face index: detectFaces, 0~detectedFaces-1
long[] faceContext = new long[1];
retCode = AMTLiveFaceService.getFaceContext(instanceContext, 0, faceContext);
if(0 == retCode )
        {
            int[] score = new int[1];
            retCode = AMTLiveFaceService.getLiveness(faceContext[0], score);
            if (0 == retCode && score[0] < 65)
        {
            //A doubtful false face
```

```
        }
    }
```

4.8 End of the Program

4.8.1 End Procedure Description

//Releases face recognition engine. For interface details, please refer to the corresponding interface description in **terminate**.

```
public static int terminate(long context);
```

Remarks

If you use an interface that corresponds to 1:N, call the interface **dbClear** to clear the 1:N cache before calling the interface **terminate**.

4.8.2 End Example Program Description

//Clears the 1:N cache (with a 1: corresponding interface), where instanceContext is the instance pointer after the algorithm initializes successfully

```
ret = AMTLiveFaceService.dbClear(instanceContext);
```

//Releases the resource, where instanceContext[0] is the instance pointer after the algorithm initialization.

```
ret = AMTLiveFaceService.terminate(instanceContext);
```

5 Common problems

5.1 License Authorization

Please get the hardware information and send it to the business to apply for a license file.

See [4.1] in detail.

6 Appendix

6.1 Appendix 1 – Error Code

As shown in the following table

Error Code	Descriptions
-1	Unknown error
0	Success
1	Insufficient memory allocation
2	Parameter error
3	Failed to allocate memory
4	Invalid handle
5	Invalid Parameter code
6	Failed to get the eyes space
7	Invalid face index number
8	The comparison score is too low
9	The actual face template length is larger than the pre-allocated face template length
10	Interface is not supported
11	Other errors
12	Invalid face ID

13	1:N identification failure, no corresponding face template was found.
14	Failed to load dynamic library
15	Image type Parameter error
16	Exceeds the maximum capacity of 1:N
17	The actual face thumbnail length is larger than that of the pre-allocated face thumbnail length

Remarks

When returns error code 11, call **GetLastError** Interface to get the error messages.

6.2 Appendix 2 – Parameter Code

Descriptions are as follows:

Parameter code	Type	Descriptions
AMTLIVEFACE_PARAMETER_SET_MIN_EYE_DIST(1005)	string	Sets the distance between the eyes. The default distance is 60
AMTLIVEFACE_PARAMETER_SET_THRESHOLD_IFCAE_VERIFY(1009)	string	Sets (get) 1:1 threshold
AMTLIVEFACE_PARAMETER_GLOBAL_LICENSE_FILENAME(1011)	string	Sets the permission file path
AMTLIVEFACE_PARAMETER_GLOBAL_LICENSE_DATA(1012)	unsigned char*	Sets license file data

Remarks

Set (get) 1:1 verification threshold, set (get) Parameter value: pure numeric string.

6.3 Appendix 3 – ICAO Feature Data

Descriptions are as follows:

Feature code	Descriptions
AMTFACE_ICAO_FEATURE_ID_AGE(0)	Age assessment
AMTFACE_ICAO_FEATURE_ID_GENDER(1)	Assessment of gender
AMTFACE_ICAO_FEATURE_QUALITY(2)	Assess the quality of the face
AMTFACE_ICAO_FEATURE_ANGLE(3)	Face angle
AMTFACE_ICAO_FEATURE_GLASSES(4）	Assessment of glasses
AMTFACE_ICAO_FEATURE_MASK(5)	Mask detection
AMTFACE_ICAO_FEATURE_HAIR(8)	Assessment of hair
AMTFACE_ICAO_FEATURE_SKIN_COLOR(9)	Assessment of skin color
AMTFACE_ICAO_FEATURE_EXP(13)	Expression

Remarks

- When obtaining face quality, score is the returned quality score from 0~100.
- When obtaining gender, score returns.

1	Male
0	Female

- When getting glasses, score returns.

1	With Glasses
0	No Glasses

- When detecting masks, score returns.

0	Wearing a mask correctly

1	Failing to wear the mask correctly
2	Not wearing a mask
3	Unknown if wearing a mask
4	Invalid

Note: the accuracy of the mask type is for reference only)

- When getting hair, score returns.

0	Bald
1	Small amount of hair
2	Short hair
3	Long hair
4	Unknown
5	Invalid

- When getting skin color, score returns.

0	yellow skin
1	white skin
2	brown skin
3	black skin
4	unknown
5	invalid

- When getting an expression, score returns.

0	unknown
1	happy
2	serious
3	surprise
4	angry
5	sad
6	neutral

6.4 Appendix 4 – Threshold Description

- 1:1 recommended threshold value is 56
- Mainly used in scenes with fuzzy image quality, such as Identity card photos and other credentials.
- Recommended 1:N threshold value, and the actual application can adjust the threshold as needed.

Database Storage Capacity	Reference Threshold
100,000	71
50,000	70
20,000	68
10,000	66
5,000	65

190 Bluegrass Valley Pkwy,

Alpharetta, GA 30005, USA

E-mail: info@armatura.us

www.armatura.us