API Development Manual:

AMTPalmMobile SDK for iOS

API Version: 1.0

Doc Version: 1.0

September 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.

Copyright © 2022 ARMATURA LLC. All rights reserved.

Without the prior written consent of ARMATURA LLC. no portion of this manual can be copied or forwarded in any way or form. All parts of this manual belong to ARMATURA and its subsidiaries (hereinafter the "Company" or "ARMATURA").

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 **AMTPalmMobile SDK for iOS**.

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

Document Conventions

Conventions used in this manual are listed below:

GUI Conventions

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

Symbols

Convention	Description
	This represents a note that needs to pay more attention to.
	The general information which helps in performing the operations faster.
	The information which is significant.
	Care taken to avoid danger or mistakes.
	The statement or event that warns of something or that serves as a cautionary example.

Table of Contents

1 Introduction

This document will provide basic SDK development guide and technical background to help application developers/integrators better understand AMTPalmMobile SDK in their development practice.

The following sections will explain all the required information on how to perform and integrate AMTPalmMobile SDK.

1.1 Overview

AMTPalmMobile biometric recognition algorithm is AI computer vision-based palm recognition algorithm on true-color RGB images. It not only recognizes and supports palm liveness detect, but also has strong adaptability to various environments of varying lighting condition. It can perform the palm recognition with accuracy even with partially captured or blurred palm images, and less impacted by ambient light. It is fully open to software developers and system integrators, and the SDK can be customized to meet the customer requirements. We keep a consistent model for palm detection, feature extraction, and matching to ensure the compatibility throughout all different SDK versions and cross various platforms.

1.2 Algorithm Features

- **1:N High-Speed Matching Algorithm**
  While maintaining the high stability in performance, the algorithm uses a multi-level comparison mode and optimized classifier parameters, to achieve high-speed matching for large-volume users.
- **Palm Quality Assessment**
  Evaluate the image quality of the target palm.
- **Highly Secure Anti-Spoofing Protection**
  Liveness detection under visible light to ensure the palm is a real and right one, protect the target application from forgery attacks.
- **High-Tolerance to Palm Postures**
  The algorithm is not only adaptable to wide Pitch (±30°), Yaw (±45°), or Roll (±30°) angels of palm postures, but also effectively identifies various palm shapes from tensed to bended.

The high posture tolerance allows user to perform palm recognition in a natural and comfortable way, which greatly improves the user experience.

1.3 Advantage of the Algorithm

- The algorithm works well with true color RGB images captured by most common digital cameras for mobile devices or web browsers.
- Simple, intuitive, and developer-friendly programming interfaces.
- Well-documented development guide on code tutorial.
- Rich programming interfaces provide value-added features on applications.

2 Technical Specifications

Development Language

This SDK provides iOS API interface and supports C++ language development.

Platform Requirements

A 64-bit iOS platform is required.

Technical Parameters

Parameter	Description
Template size	544B
Posture adaptability	Yaw ≤45°, Pitch ≤30°, Roll ≤90°, Bend≤20°
Palm detection	< 15 ms
Palm feature extraction	< 45 ms
Palm verification/identification (1:10,000)	< 10 ms
Number of palm templates supported	5000
Accuracy	TAR=98.2% when FAR=0.05%

The above performance is based on the tests conducted with the following specifications: Image resolution: 640x640, CPU: Intel(R) Core(TM) i5-9400 CPU @ 2.90Ghz, RAM: 16GB.

2.1 Architecture

2.1.1 SDK File

- Copy the following file to your MacOS.

File Name	Description
AMTPalmMobile.dylib	Biometric Interface Dynamic-link Library

2.1.2 Development Setup

SDK Dynamic-link library files can be copy-paste and installed directly

Please make sure your operating system and computer configuration meet the requirements of software operation before installing the AMTPalmMobile SDK.

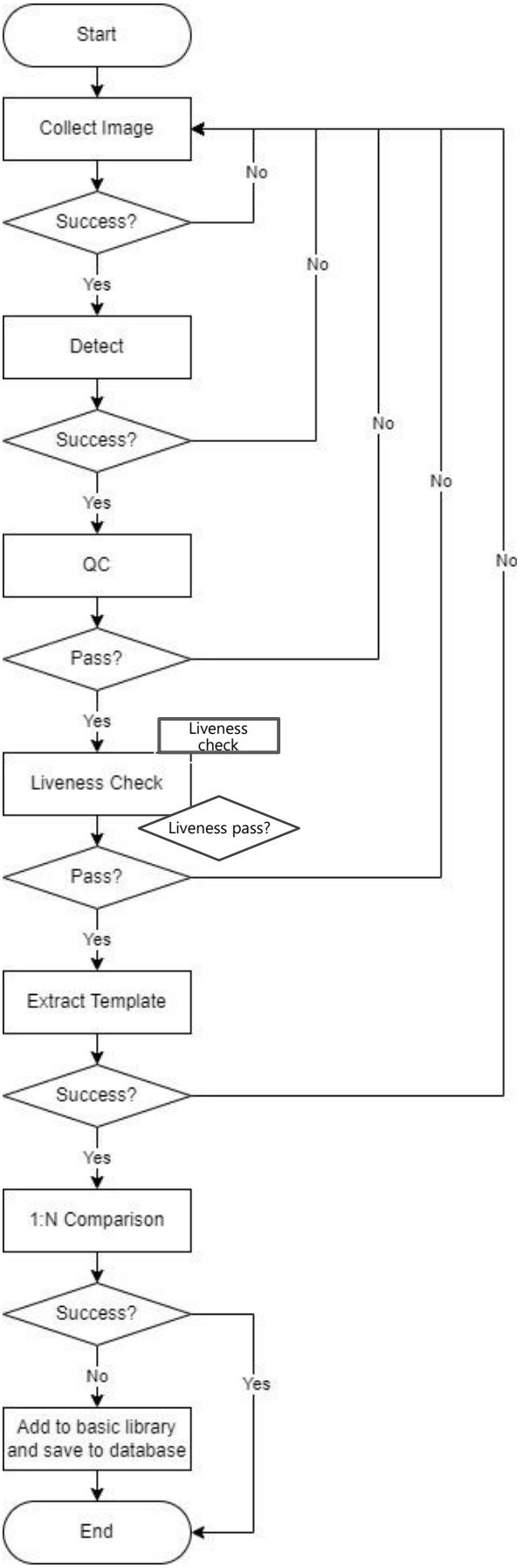
Copy AMTPalmMobile.dylib and related files in AMTPalmMobile SDK to the path specified by the user.

2.2 Programming Guide

The following sections will provide introduction and walk-through of the key operating processes and the Biometric registration/comparison processes of the algorithms in AMTPalmMobile SDK for the purpose of further understanding and development.

2.2.1 Registration Process

The extracted palm information can be directly used as the registration template during palm registration. Refer to Section 3 SDK Interface Specification for more information.

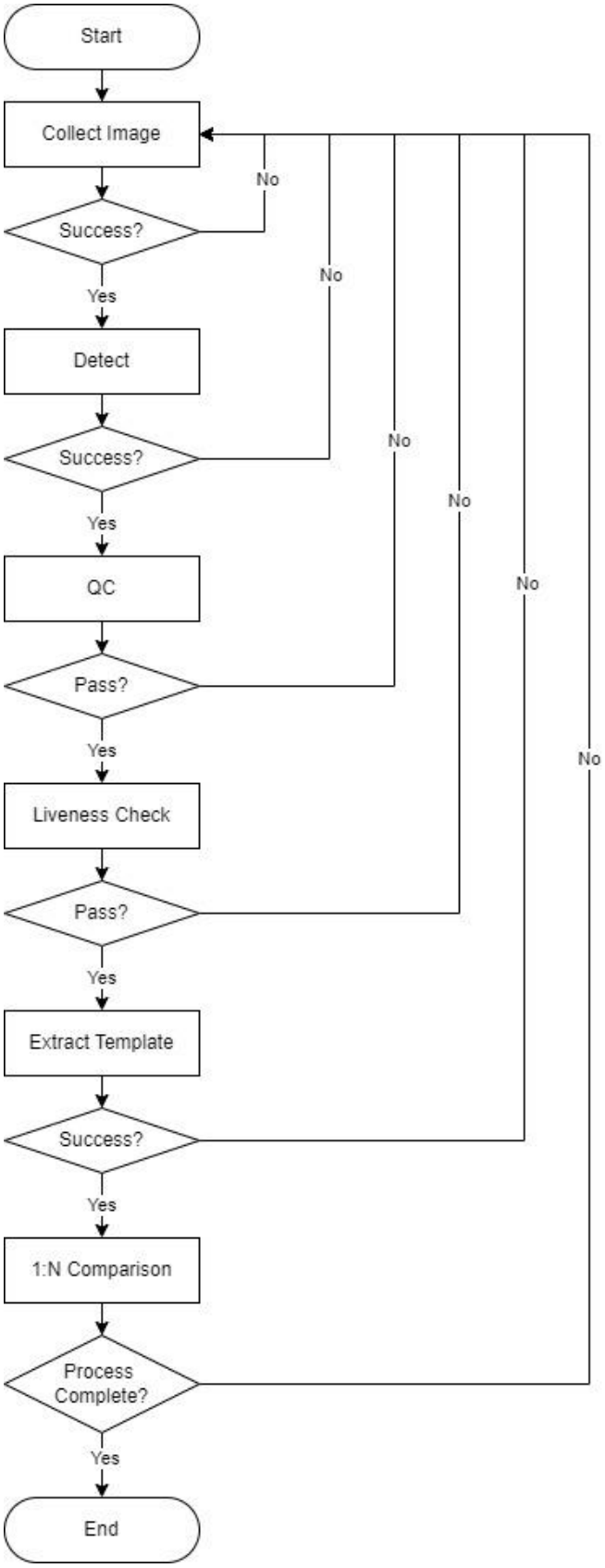

Start
Collect Image
No
Success?
Yes
Detect
No
Success?
Yes
QC
No
Pass?
Yes
Liveness check
Liveness Check
Liveness pass?
No
Pass?
Yes
Extract Template
No
Success?
Yes
1:N Comparison
Success?
No
Yes
Add to basic library and save to database
End

Process Description:

- Call the enrollment class to collect images
- Once the image is successful collected, call AMTPalmMobile_Detect to detect the palm
- After the palm image is successfully detected, call AMTPalmMobile_GetPalmQuality for quality inspection.
- After passing quality inspection, call AMTPalmMobile_GetPalmVL for liveness detection.
- If pass liveness detection (over the liveness threshold value), call AMTPalmMobile_GetTemplate to extract the palm template
- After the template is successfully extracted, call AMTPalmMobile_DBIdentify to perform 1:N matching to check whether the current template has been registered or not. If it has been registered before, it will prompt the user that the palm has been registered and stops the registration process.
- If the 1:N matching returns with negative value, means no template matched from the database, call AMTPalmMobile_DBSet to add the palm template to the base library (cache) and save the palm template to the database
- Complete the registration process

2.2.2 Verification/Identification Process

For palm identification (1:N matching), all registered templates need to be added to base library (cache) first. It is recommended to call AMTPalmMobile_DBSet and add all palm template to the base library after the algorithm is successfully initialized. This process is also recommended for palm verification (1:1 matching).


Start
Collect Image
Success?
No
Yes
Detect
Success?
No
Yes
QC
Pass?
No
Yes
Liveness Check
Pass?
No
Yes
Extract Template
Success?
No
Yes
1:N Comparison
Process Complete?
No
Yes
End

Process Description:

- Call the enrollment class to collect images
- Once the image is successful collected, call AMTPalmMobile_Detect to detect the palm
- After the palm is detected, call AMTPalmMobile_GetPalmQuality for quality check.
- After passing quality check, call AMTPalmMobile_GetPalmVL for liveness detection.
- If liveness detection returns with positive value, call AMTPalmMobile_GetTemplate to extract the template
- Call AMTPalmMobile_DBIdentify to perform 1:N matching to complete the process.

3 SDK Interface Description

3.1 Template Format

Template Type	Data Length	Description
Palm template	544 Bytes	Work as registration template or verification/ identification template

3.2 Interface Description

AMTPalmMobile.dylib Dynamic-link library is a dynamic library for biometric interface. It is mainly used for palm detection, template extraction, registration, comparison, and palm specification.

3.2.1 AMTPalmMobile.dylib

3.2.1.1 Function List

Interface	Description
AMTPalmMobile_Version	Get AMPalmMobile SDK version
AMTPalmMobile_Init	Initialize algorithm resources
AMTPalmMobile_Final	Clear algorithm cache
AMTPalmMobile_GetParam	Get detection parameters
AMTPalmMobile_LoadModels	Load models from disk into memory
AMTPalmMobile_LoadModelsConfig	Load templates from disk into memory based on configuration
AMTPalmMobile_Detect	Palm Detection
AMTPalmMobile_Detect2Img	Palm Detection from mosaic images
AMTPalmMobile_DetectRotation	Palm rotation Detection
AMTPalmMobile_DetectFile	Palm detection from image files

AMTPalmMobile_GetObject	Get the palm information struct
AMTPalmMobile_GetNormalizedImage	Get Normalized palm image
AMTPalmMobile_SaveNormalizedImage	Save Normalized palm image
AMTPalmMobile_GetFeature	Extracts palm features
AMTPalmMobile_SetParam	Set palm detection parameters
AMTPalmMobile_GetTemplate	Extracts palm template
AMTPalmMobile_FreeTemplate	Release extracted palm template
AMTPalmMobile_GetTemplateSize	Calculate size of palm template
AMTPalmMobile_Verify	Perform 1:1 matching
AMTPalmMobile_GetPalmVL	Palm liveness check under visible light
AMTPalmMobile_DBOpen	Connect database
AMTPalmMobile_DBClose	Close database
AMTPalmMobile_DBSet	Store the original template data in the database
AMTPalmMobile_DBDel	Delete specific template from the template database
AMTPalmMobile_DBGet	Get specific template from the template database
AMTPalmMobile_DBCountByID	Calculate the total number of original palm templates for the specific ID(s)
AMTPalmMobile_DBCountID	Calculate the total number of ID assigned in database
AMTPalmMobile_DBIdentify	Identify in database
AMTPalmMobile_DBReset	Clears all data in database
AMTPalmMobile_DBVerify	Performs a 1:1 matching between specified templates
AMTPalmMobile_DBBegin	Database operations: (BEGIN TRANSACTION)
AMTPalmMobile_DBCommit	Database operations: (COMMIT) Save changes

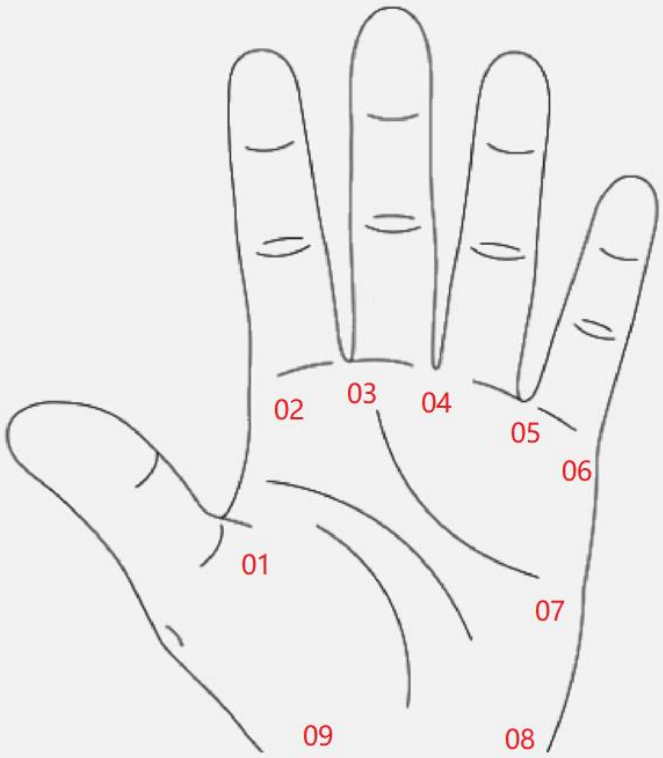
AMTPalmMobile_DBRollback	Database operations: (ROLLBACK) undo the previous changes
AMTPalmMobile_DBForAll	Iterating over the entire database
TCallbackFun	Callback function for iterating over the user ID
AMTPalmMobile_GetPalmQuality	Get palm quality

3.2.1.2 Description of the structure

```
// Struct for feature points on target image
typedef struct Landmark {
    float x, y;   //the location of a landmark (feature point coordinates on target image)
} TLandmark;

//Struct for target rectangular frame
typedef struct Box {
    float x, y, width, height;  //The location and sizes of bound box. (x,y) is the coordinates of the upper left corner of the target rectangular frame, width is the width of frame, and height is the height of frame.
} TBox;

// Struct for target image information
typedef struct TObject {
    int class_id;  //A number indicates what is it (Identify target image)
    TBox box;    //The bound box of the object (target rectangular frame)
    TLandmark *landmarks;   //The landmarks of an object (target feature point)
    int landmark_count;    //The count of landmarks for an object (the number of coordinates of target feature point)
    float yaw, roll, pitch;  //The 3D posture of the object (the 3D posture of target palm)
    float score;    // The accuracy score of the object (score for accuracy)
    int pic;
} TObject;

// Palm feature point coordinates
/**************************************************************************
```

/* Landmark[9]

*



* Note: The feature points will be marked clockwise in consecutive order for both left and right hands, ignore the position of thumb.

#define PARAM_IN //Indicates that the corresponding parameter is an input parameter

#define PARAM_OUT //Indicates that the corresponding parameter is an output parameter

#define PARAM_INOUT //Indicates that the corresponding parameter is an input/output parameter

3.2.1.3 AMTPalmMobile_Version

Function Syntax

const char * APICALL AMTPalmMobile_Version(PARAM_OUT int *version, PARAM_OUT int *major, PARAM_OUT int *minor);

Description

Request SDK version.

Parameters

Parameter	Description
version[out]	Return version number
major[out]	Return major version number
minor[out]	Return minor version number

Returns

SDK version number string

3.2.1.4 AMTPalmMobile_init

Function Syntax

void* APICALL AMTPalmMobile_Init();

Description

Initializing algorithmic resources.

Parameters

None

Returns

Return void pointer

Remarks

1. The above interface should be successfully called before calling any other interface.
2. The algorithm resource will only need to be initialized once during the entire program cycle.

3.2.1.5 AMTPalmMobile_Final

Function Syntax

void APICALL AMTPalmMobile_Final(PARAM_IN void * handle);

Description

Free up algorithm resources.

Parameters

AMTPalmMobile_Init Return void pointer.

Returns

None

Remarks

1. The above interface should be called to release algorithm resource when terminating the program.

3.2.1.6 AMTPalmMobile_GetParam

Function Syntax

int APICALL AMTPalmMobile_GetParam(PARAM_IN void *handle, PARAM_IN const char *item, PARAM_OUT char *value, PARAM_IN int size);

Description

Get target detection parameters. (Target here means palm).

Parameters

Parameter	Description
handle[in]	Call void pointer
item[in]	Available parameters include: "min_size" => Minimum target pixel value "max_count" => Maximum number of detected targets "threshold" => threshold of the score earned by the first target image "threshold2" => Thresholds of scores earned by other targets "width" => Width of target image, which also determines detection speed/accuracy. Recommended value 320/480/640 "height" => Height of target image, which also determines

	detection speed/accuracy. Recommended value 320/480/640
value[out]	Pre-assigned byte string
size[in]	Length of pre-assigned byte string

Returns

Success when return value equals or larger than zero, error when return value smaller than zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.7 AMTPalmMobile_LoadModels

Function Syntax

int APICALL AMTPalmMobile_LoadModels(PARAM_IN void *handle);

Description

Load model from disk into memory, normally takes a while to process.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals to zero, error when return value smaller than zero. See zero, See error code for more detail (Appendix 1: Error Code)

Remarks

1. Call the above code after initializing with AMTPalmMobile_Init. Both of them should be successfully called before calling any other interface.

3.2.1.8 AMTPalmMobile_LoadModelsConfig

Function Syntax

int APICALL AMTPalmMobile_LoadModelsConfig(PARAM_IN void *handle, PARAM_IN const char* configFile);

Description

Load model from disk into memory, normally takes a while to process.

Parameters

Parameter	Description
handle[in]	Call void pointer
configFile[in]	Configuration file path, this value should always be a non-empty string

Returns

Success when return value equals to zero, error when return value smaller than zero. See zero, see error code for more detail (Appendix 1: Error Code)

Remarks

1, Call the above code, or AMTPalmMobile_LoadModelsConfig after initializing with AMTPalmMobile_Init. Both of them should be successfully called before calling any other interface.

3.2.1.9 AMTPalmMobile_Detect

Function Syntax

int APICALL AMTPalmMobile_Detect(PARAM_IN void * handle, PARAM_IN const unsigned char *image, PARAM_IN int width, PARAM_IN int height, PARAM_IN const char *image_format);

Description

Detect target from images, The return value will be the number of detected targets. (Target here means palm).

Parameters

Parameter	Description
handle[in]	Call void pointer
image[in]	Input images need to be converted in to chart with specified format
width[in]	Image width

height[in]	Image height
image_format[in]	Supported image format include: "rgb888", "bgr888", "rgba8888", "bgra8888", "bgr565", "nv21", "gray"

Returns

Success when the returned value is larger than or equal to zero (the number of target detected), error if less than zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.10 AMTPalmMobile_Detect2Img

Function Syntax

int APICALL AMTPalmMobile_Detect2Img(PARAM_IN void * handle, PARAM_IN const unsigned char *image1, PARAM_IN const unsigned char *image2, PARAM_IN int width, PARAM_IN int height, PARAM_IN const char *image_format);

Description

Detect target from mosaic images. The return value will be the number of targets detected. The detected images will be merged automatically as: When height > width, images will be merged in the following order: (image1) on the left, (image2) on the right; When width > height (image1) will be on top and (image2) on the bottom. (Target here means palm).

Parameters

Parameter	Description
handle[in]	Call void pointer
Image1[in]	Image one chart with specified format
Image2[in]	Image two chart with specified format
width[in]	Image width, the length and width of the two images should be aligned
height[in]	Image height, the length and width of the two images should be aligned
image_format[in]	Supported image format include: "rgb888", "bgr888", "rgba8888", "bgra8888", "bgr565", "nv21", "gray"

Returns

Success when the returned value is larger than or equal to zero (the number of target detected), error if less than zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.11 AMTPalmMobile_DetectRotation

Function Syntax

int APICALL AMTPalmMobile_DetectRotation(PARAM_IN void *handle,

PARAM_IN const unsigned char *image,

PARAM_IN int width,

PARAM_IN int height,

PARAM_IN const char *image_format,

PARAM_IN int angle,

PARAM_IN int flipx,

PARAM_IN int flipy);

Description

Detect target from images, The return value will be the number of targets detected. Support image rotation to fix the intended orientation of input image. (target here means palm)

Parameters

Parameter	Description
handle[in]	Call void pointer
Image[in]	Input images need to be converted in to chart with specified format
width[in]	Image width
height[in]	Image height
image_format[in]	Supported image format include: "rgb888", "bgr888", "rgba8888", "bgra8888", "bgr565", "nv21", "gray"
angle[in]	The available rotation angles include: 0, 90, 180, 270
flipx[in]	Input whether to flip vertically on the x-axis. Image will not be flipped when 0 is entered as input value.
flipy[in]	Input whether to flip horizontally on the y-axis. Image will not be flipped when 0 is entered as input value.

Returns

Success when the returned value is larger than or equal to zero (the number of target detected), error if smaller than zero. See error code for more detail (Appendix 1: Error Code).

3.2.1.12 AMTPalmMobile_DetectFile

Function Syntax

int APICALL AMTPalmMobile_DetectFile(PARAM_IN void * handle, PARAM_IN const char *image_filename);

Description

Detect target from files, The return value is the number of targets detected (target here means palm).

Parameters

Parameter	Description
handle[in]	Call void pointer
image_filename[in]	image file name

Returns

Success when the returned value is larger than or equal to zero (the number of target detected), error if less than zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.13 AMTPalmMobile_GetObject

Function Syntax

int APICALL AMTPalmMobile_GetObject(PARAM_IN void * handle,PARAM_IN int index, PARAM_OUT TObject **obj);

Description

Get the target information structure

Parameters

Parameter	Description
handle[in]	Call void pointer

index[in]	Target index, less than the number of detected targets (0 ~ number of detected targets - 1)
obj[out]	Returns the target structure pointer (See 3.2.1.2. Structure Description for more detail)

Returns

Success when return value equals zero, error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.14 AMTPalmMobile_GetNormalizedImage

Function Syntax

int APICALL AMTPalmMobile_GetNormalizedImage(PARAM_IN void * handle, PARAM_IN int index, PARAM_IN const char *name, PARAM_OUT unsigned char **image_buffer, PARAM_OUT int *image_height, PARAM_OUT int *image_width);

Description

Get the normalized image of the target.

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]	Target index, less than the number of detected targets (0 ~ number of detected targets - 1)
name[in]	Type of Normalization
image_buffer[out]	Normalization pointer
image_height[out]	Normalized height
image_width[out]	Normalized width

Returns

Success when return value equals zero, error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.15 AMTPalmMobile_SaveNormalizedImage

Function Syntax

int APICALL AMTPalmMobile_SaveNormalizedImage(PARAM_IN void * handle,

PARAM_IN int index, PARAM_IN const char *name, PARAM_IN const char *image_filename);

Description

Save the normalized image of the target. (Target here means palm)

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]	Target index, less than the number of detected targets (0 ~ number of detected targets - 1)
name[in]	Type of Normalization
image_filename[in]	Image file name

Returns

Success when return value equals zero, error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.16 AMTPalmMobile_GetFeature

Function Syntax

int APICALL AMTPalmMobile_GetFeature(PARAM_IN void * handle, PARAM_IN int index, PARAM_IN const char *name, PARAM_OUT float *values, PARAM_IN int count);

Description

Extracts the features of the target. (Target here means palm).

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]]	Target index, less than the number of detected targets (0 ~ number of detected targets-1)
name[in]	Type of feature
values[out]	Value of feature
count[out]	Length of feature

Returns

Success when return value equals or larger than zero, error when return value smaller than

zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.17 AMTPalmMobile_SetParam

Function Syntax

int APICALL AMTPalmMobile_SetParam(PARAM_IN void *handle, PARAM_IN const char *item, PARAM_IN const char *value);

Description

Set detection parameters.

Parameters

Parameter	Description
handle[in]	Call void pointer
item[in]	Available parameters include: "min_size" => Minimum target pixel value "max_count" => Maximum number of detected targets "threshold" => threshold of the score earned by the first target image "threshold2" => Thresholds of scores earned by other targets "width" => Width of target image, which also determines detection speed/accuracy. Recommended value 320/480/640 "height" => Height of target image, which also determines detection speed/accuracy. Recommended value 320/480/640
value[in]	Parameter string
values[out]	Value of feature
count[out]	Length of feature

Returns

Success when return value equals or larger than zero, error when return value smaller than zero. See error code for more detail (Appendix 1: Error Code)

3.2.1.18 AMTPalmMobile_GetTemplate

Function Syntax

int APICALL AMTPalmMobile_GetTemplate(PARAM_IN void * handle, PARAM_IN int index, PARAM_OUT unsigned char **temp);

Description

Extract feature template.

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]	Target index, less than the number of detected targets (0 ~ number of detected targets-1)
temp[out]	Returns template pointer accordingly

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

[Note]

1. When not using feature template data, call AMTPalmMobile_FreeTemplate (See 3.2.1.27 Interface Description for more detail) to free up feature template.

3.2.1.19 AMTPalmMobile_ExtractTemplate

Function Syntax

int APICALL AMTPalmMobile_ExtractTemplate(PARAM_IN void * handle, PARAM_IN const unsigned char* input, PARAM_IN const int height, PARAM_IN const int width, PARAM_IN const char* input_format, PARAM_IN const int cls, PARAM_OUT unsigned char **temp);

Description

Extract template.

Parameters

Parameter	Description
handle[in]	Call void pointer
input[in]	Normalized image
height[in]	the height of the normalized image
width[in]	the width of the normalized image

input_format[in]	The format of the normalized image
cls[in]	the class of the normalized image
temp[out]	Return template list pointer, need FreeTemplate to release interface

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

[Note]

1. When not using feature template data, call AMTPalmMobile_FreeTemplate (See 3.2.1.27 Interface Description for more detail) to free up feature template.

3.2.1.20 AMTPalmMobile_FreeTemplate

Function Syntax

AMTAPI void APICALL AMTPalmMobile_FreeTemplate(PARAM_IN unsigned char *temp);

Description

Release the extracted feature template.

Parameters

Parameter	Description
temp[in]	template pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.21 AMTPalmMobile_GetTemplateSize

Function Syntax

AMTAPI int APICALL AMTPalmMobile_GetTemplateSize(PARAM_IN void *handle,

PARAM_IN const unsigned char *temp);

Description

Calculate the size of feature template.

Parameters

Parameter	Description
handle[in]	Call void pointer
temp[in]	template pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.22 AMTPalmMobile_Verify

Function Syntax

AMTAPI int APICALL AMTPalmMobile_Verify(PARAM_IN void * handle,PARAM_IN const unsigned char *template1,PARAM_IN const unsigned char *template2,PARAM_OUT float *similarity_score)

Description

Compares two templates, returns a similarity score between 0~99.3799.

Parameters

Parameter	Description
handle[in]	Call void pointer
template1[in]	Input template1
Template2[in]	Input template2
similarity[out]	Return the similarity scores for pairs of templates

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.23 AMTPalmMobile_GetPalmVL

Function Syntax

AMTAPI int APICALL AMTPalmMobile_GetPalmVL(PARAM_IN void * handle,PARAM_IN int index,PARAM_OUT float *values);

Description

Palm liveness check under visible light.

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]	Palm Index
values[out]	Palm score

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.24 AMTPalmMobile_DBOpen

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBOpen(PARAM_IN void *handle,PARAM_IN const char *db);

Description

Access database.

Parameters

Parameter	Description
handle[in]	Call void pointer
db[in]	Name of database

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.25 AMTPalmMobile_DBClose

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBClose(PARAM_IN void *handle);

Description

Close the database.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.26 AMTPalmMobile_DBSet

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBSet(PARAM_IN void *handle,

PARAM_IN const char *id,

PARAM_IN const unsigned char **sampleTemplates,

PARAM_IN int count,

PARAM_IN AMTModal m);

Description

Store the original template data in the database. Note: When return value is zero the old database will be switched to new database.

Parameters

Parameter	Description
handle[in]	Call void pointer
id[in]	Template ID
sampleTemplates[in]	Multi-template samples
count[in]	Number of templates
m[in]	Identify modal (palm)

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.27 AMTPalmMobile_DBDel

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBDel(PARAM_IN void *handle, PARAM_IN const char *id);

Description

Delete specific template from database based on template ID.

Parameters

Parameter	Description
handle[in]	Call void pointer
id[in]	Target id

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.28 AMTPalmMobile_DBGet

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBGet(PARAM_IN void *handle,

PARAM_IN const char *id,

PARAM_OUT unsigned char *sampleTemplates,

PARAM_IN int size,

PARAM_IN AMTModal m);

Description

Read the specified palm template from the template database and place in sampleTemplates in proper order.

The size of each template will be calculated by AMTPalmMobile_GetTemplateSize(handle, NULL). This function will return the number of valid templates.

Parameters

Parameter	Description
handle[in]	Call void pointer
id[in]	Enter ID
sampleTempalte[out]	Output Template
size[in]	Size of each template
m[in]	Identify modal (palm)

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.29 AMTPalmMobile_DBCountByID

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBCountByID(PARAM_IN void *handle, PARAM_IN const char *id, PARAM_IN AMTModal m);

Description

Calculates the number of original palm templates for specific ID.

Parameters

Parameter	Description
handle[in]	Call void pointer
id[in]	Enter id
m[in]	Identify modal (palm)

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.30 AMTPalmMobile_DBCountID

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBCountID(PARAM_IN void *handle);

Description

Calculate the total number of ID stored in database.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.31 AMTPalmMobile_DBIdentify

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBIdentify(PARAM_IN void *handle,

PARAM_IN const unsigned char *liveTemplate,

PARAM_OUT char *id,

PARAM_OUT float *similarity_score,

PARAM_IN float minScore,

PARAM_IN float maxScore,

PARAM_IN AMTModal m);

Description

Similarity score identification

Parameters

Parameter	Description
handle[in]	Call void pointer
liveTemplate[in]	Target template pointer
id[out]	Output ID
similarity_score[out]	Output similarity score
minScore[in]	minimum score
maxScore[in]	maximum score
m[in]	Identify modal (palm)

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.32 AMTPalmMobile_DBReset

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBReset(PARAM_IN void *handle);

Description

Clear all data in database.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.33 AMTPalmMobile_DBVerify

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBVerify(PARAM_IN void *handle,

PARAM_IN const unsigned char *liveTemplate,

PARAM_IN const char *id,

PARAM_OUT float *similarity_score,

PARAM_IN AMTModal m);

Description

Perform 1 on 1 comparison between specified templates, return similarity scores between 0~99.3799.

Parameters

Parameter	Description
handle[in]	Call void pointer
liveTempalte[in]	Template to be compared
id[in]	ID to be compared
similarity_score[out]	Similarity score
m[in]	Identify modal (palm)

Returns

Greater than or equal to zero means success, less than zero means error (see details in Appendix 1: Error Code)

3.2.1.34 AMTPalmMobile_DBBegin

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBBegin(PARAM_IN void *handle);

Description

Database operation: BEGIN TRANSACTION, start transaction.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.35 AMTPalmMobile_DBCommit

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBCommit(PARAM_IN void *handle);

Description

Database operation: COMMIT, save changes.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.36 AMTPalmMobile_DBRollback

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBRollback(PARAM_IN void *handle);

Description

Database operation: ROLLBACK, undo changes.

Parameters

Parameter	Description
handle[in]	Call void pointer

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.37 AMTPalmMobile_DBForAll

Function Syntax

AMTAPI int APICALL AMTPalmMobile_DBForAll(PARAM_IN void *handle, PARAM_IN TCallbackFun f, PARAM_INOUT void *param);

Description

Iterate over the entire database.

Parameters

Parameter	Description
handle[in]	Call void pointer
f[in]	Callback function
param	Callback function parameters

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.38 TCallbackFun

Function Syntax

typedef int (APICALL * TCallbackFun)(PARAM_IN void *handle, PARAM_IN int index, PARAM_IN const char *id);

Description

Callback function to iterate over the user ID. ID means user number, index is not used yet.

Parameters

Parameter	Description
handle[in]	Call void pointer
id[in]	Enter ID
index[in]	Enter 0

Returns

Success when return value equals or larger than zero (the length of template), error when return value smaller than zero, see error code for more detail (Appendix 1: Error Code)

3.2.1.39 AMTPalmMobile_GetPalmQuality

Function Syntax

AMTAPI int APICALL AMTPalmMobile_GetPalmQuality(PARAM_IN void * handle, PARAM_IN int index, PARAM_OUT float *values);

Description

Get palm quality.

Parameters

Parameter	Description
handle[in]	Call void pointer
index[in]	Target algorithm index
values[in]	Quality Score Storage Pointer

Returns

Greater than or equal to zero means success, less than zero means error (see details in Appendix 1: Error Code)

Appendix

Appendix 1: Error Code

Error Code	Description
0	Call succeeded
-1000	Certificate error
-1001	Error reading configuration file
-1002	The feature name is wrong, or the feature is not supported
-1003	Model name error or such model is not supported
-1004	Error identifying ROI (region of interest) name, or the ROI is not supported
-1005	The normalization name is wrong, or the normalization is not supported
-1006	Null pointer error
-1007	Target not detected
-1008	Target index exceeded error
-1009	Input greater than space of temp cache location
-1010	Input parameter error
-1011	Configuration parameter keyword error
-1012	Configuration parameter value error
-1013	Feature type error
-1014	Model type error
-1015	Normalization type error
-10001	Invalid template
-10002	Failed to connect to database or database creation failed
-10003	Failed to access database
-10004	Database access error
-10005	Template size error
-10006	ID not found in database

Appendix 2: Glossary

The following definitions will help our users understand the common functions of biometric identification applications when developing the biometric identification applications.

Verification/Identification template

Verification/Identification templates are used to either 1:1 verification or 1:N identification. The palm templates are obtained by calling the AMTPalmMobile_GetTemplate interface.

Registration template

Registration templates are used to registration that is added to the basic library (cache). A registration template is the palm templates returned by calling the **AMTPalmMobile_GetTemplate** interface.

Palm Registration

The palm collecting device captures a palm image and then extracts palm template, which is transferred to the backend and stored in database as a registered palm for later palm comparison.

Palm Verification (1:1)

1:1 verification is a process of verifying whether a user has a valid identity based on the user ID and palm template or determining whether the registered template and the verification templates extracted matches the same captured palm image.

That is, 1:1 biometric verification process authenticates a person's identity by comparing the captured biometric template with a biometric template of that person pre-stored in the database.

Palm Identification (1:N)

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

So thus, A one-to-many (1:N) biometric identification process instantly compares the person's captured biometric template against ALL stored biometric templates in the system.

Appendix 3: Image backup during registration process

It is recommended to store the image used during registration process. The features may need to be re-extracted when the algorithm model is upgraded

190 Bluegrass Valley Pkwy,

Alpharetta, GA 30005, USA

E-mail: info@armatura.us

www.armatura.us

Copyright © 2022 ARMATURA LLC. All Rights Reserved.