

BioMini SDK for Linux

v3.6.0



Table of Contents

1. Introduction	1
1.1 What's New	1
1.2 History	1
1.3 Overview	2
SDK Structure	2
SDK package consists of	3
System Requirements	3
2. Getting Started	4
2.1 Driver Installation	4
SFR400 (BioMini)	4
SFR500 (BioMini Plus)	6
SFR410 (BioMini ver.2)	8
SFR600 (BioMini Slim)	9
SFR550 (BioMini Plus2)	10
2.2 BioMini Demo Program	11
3. C/C++ Development	13
3.1 Enrollment Tutorial	13
Workflow	13
Example	13
Enrollment code	16
3.2 Verification Tutorial	19
Verification Workflow	19
Example	19
verification code	22
3.3 Identification Tutorial	27
Identification Workflow	27
Example	27
Identification code	30
3.4 C/C++ APIs	35
UFS_Init	40
UFS_Update	41
UFS_Uninit	42
UFS_SetScannerCallback	43
UFS_RemoveScannerCallback	44
UFS_GetScannerNumber	45
UFS_GetScannerHandle	46
UFS_GetScannerHandleByID	47
UFS_GetScannerIndex	48
UFS_GetScannerID	49
UFS_GetScannerType	50
UFS_GetParameter	51

UFS_SetParameter	53
UFS_IsSensorOn	55
UFS_IsFingerOn	56
UFS_CaptureSingleImage	57
UFS_StartCapturing	58
UFS_StartAutoCapture	59
UFS_IsCapturing	60
UFS_AbortCapturing	61
UFS_Extract	62
UFS_ExtractEx	64
UFS_SetEncryptionKey	66
UFS_EncryptTemplate	67
UFS_DecryptTemplate	69
UFS_GetCaptureImageBufferInfo	71
UFS_GetCaptureImageBuffer	72
UFS_GetCaptureImageBufferToBMPIImageBuffer	74
UFS_GetCaptureImageBufferTo19794_4ImageBuffer	76
UFS_GetCaptureImageBufferToWSQImageBuffer	78
UFS_GetCaptureImageBufferToWSQImageBufferVar	80
UFS-DecompressWSQBMP	82
UFS-DecompressWSQBMPMem	83
UFS_DrawCaptureImageBuffer	85
UFS_DrawFeatureBuffer	87
UFS_SaveCaptureImageBufferToBMP	89
UFS_SaveCaptureImageBufferTo19794_4	90
UFS_SaveCaptureImageBufferToWSQ	91
UFS_SaveCaptureImageBufferToWSQVar	92
UFS_ClearCaptureImageBuffer	93
UFS_GetErrorString	94
UFS_GetTemplateType	95
UFS_SetTemplateType	96
UFS_SelectTemplate	97
UFS_SelectTemplateEx	99
UFS_GetFPQuality	101
UFS_GetFeatureNumber	102
UFS_EnrollUI	103
UFS_VerifyUI	105
UFS_CaptureSingleUI	107
UFM_Create	108
UFM_Delete	109
UFM_GetParameter	110
UFM_SetParameter	111
UFM_Verify	112
UFM_VerifyEx	114
UFM_Identify, UFM_IdentifyMT	116
UFM_AbortIdentify	119
UFM_IdentifyInit	120
UFM_IdentifyNext	121
UFM_RotateTemplate	123
UFM_GetErrorString	124
UFM_GetTemplateType	125

UFM_SetTemplateType	126
UFS_STATUS	127
UFM_STATUS	128

4. Java Development 129

4.1 Java Installation guide 129

4.2 Java APIs 130

UFS_Init	134
UFS_Update	135
UFS_Uninit	136
UFS_SetScannerCallback	137
UFS_RemoveScannerCallback	138
UFS_GetScannerNumber	139
UFS_GetScannerHandle	140
UFS_GetScannerHandleByID	141
UFS_GetScannerIndex	142
UFS_GetScannerID	143
UFS_GetScannerType	144
UFS_GetParameter	145
UFS_SetParameter	147
UFS_IsSensorOn	149
UFS_IsFingerOn	150
UFS_CaptureSingleImage	151
UFS_StartCapturing	152
UFS_StartAutoCapture	153
UFS_IsCapturing	154
UFS_AbortCapturing	155
UFS_Extract	156
UFS_ExtractEx	157
UFS_SetEncryptionKey	159
UFS_EncryptTemplate	160
UFS_DecryptTemplate	161
UFS_GetCaptureImageBufferInfo	162
UFS_GetCaptureImageBuffer	163
UFS_GetCaptureImageBufferToBMPImageBuffer	164
UFS_GetCaptureImageBufferTo19794_4ImageBuffer	165
UFS_GetCaptureImageBufferToWSQImageBuffer	166
UFS_GetCaptureImageBufferToWSQImageBufferVar	167
UFS-DecompressWSQBMP	168
UFS-DecompressWSQBMPMem	169
UFS_SaveCaptureImageBufferToBMP	170
UFS_SaveCaptureImageBufferTo19794_4	171
UFS_SaveCaptureImageBufferToWSQ	172
UFS_SaveCaptureImageBufferToWSQVar	173
UFS_ClearCaptureImageBuffer	174
UFS_GetErrorString	175
UFS_GetTemplateType	176
UFS_SetTemplateType	177
UFS_SelectTemplate	178
UFS_SelectTemplateEx	179

UFS_GetFPQuality	180
UFS_GetFeatureNumber	181
UFM_Create	182
UFM_Delete	183
UFM_GetParameter	184
UFM_SetParameter	186
UFM_Verify	188
UFM_Identify, UFM_IdentifyMT	190
UFM_AbortIdentify	191
UFM_IdentifyInit	192
UFM_IdentifyNext	193
UFM_RotateTemplate	195
UFM_GetErrorString	196
UFM_GetTemplateType	197
UFM_SetTemplateType	198
UFS_STATUS	199
UFM_STATUS	200

5. Appendix 202

What is Biometrics? 202

Scanners 206

BioMini Plus2	206
BioMini Slim	207
BioMini Combo	208
BioMini Plus	209
BioMini	210

1. Introduction

The BioMini SDK provides the specific functions to capture fingerprints, to display scanner real-time images on the screen while the finger is on the platen and includes all PC interfaces and drivers.

1.1 What's New

Version 3.6

- Compatible with new Suprema fingerprint scanners
 - BioMini Plus2 (SFR550)
- UFSscanner - UFS_CaptureSingleUI function added
 - Performs same as UFS_CaptureSingle and Popup Window appears
- UFMatcher - UFM_VerifyEx function added
 - Performs same as UFM_Verify, and returns matching score by 6th parameter

1.2 History

Version 3.5.5

- Do not check license file(UFLicense.dat file removed)
- UFSscanner - New APIs added
 - Save WSQ Image Buffer to variable size
 - Decompress WSQ format data to data of Bmp format
 - Get quality information of image(UFS_GetFPQuality)
 - Added EnrollUI, VerifyUI APIs

Version 3.5.0

- Compatible with new Suprema fingerprint scanners
 - BioMini Slim (SFR600), BioMini Combo and SFU-S20
- UFSscanner - UFS_StartAutoCapture added
 - Auto finger detect and capture function
- UFSscanner - UFS_ExtractEx function added
 - Template extraction function for large size template
- UFSscanner - UFS_SelectTemplateEx added
 - Template selection function for large size template

Version 3.4.0

- Universal manufacturer ID embedded in scanners without license control
- Support new sensor, which has AGC(Auto Gain Control) feature on BioMini

- Removed brightness setting from new sensor since the AGC automatically finds it's optimal brightness setting

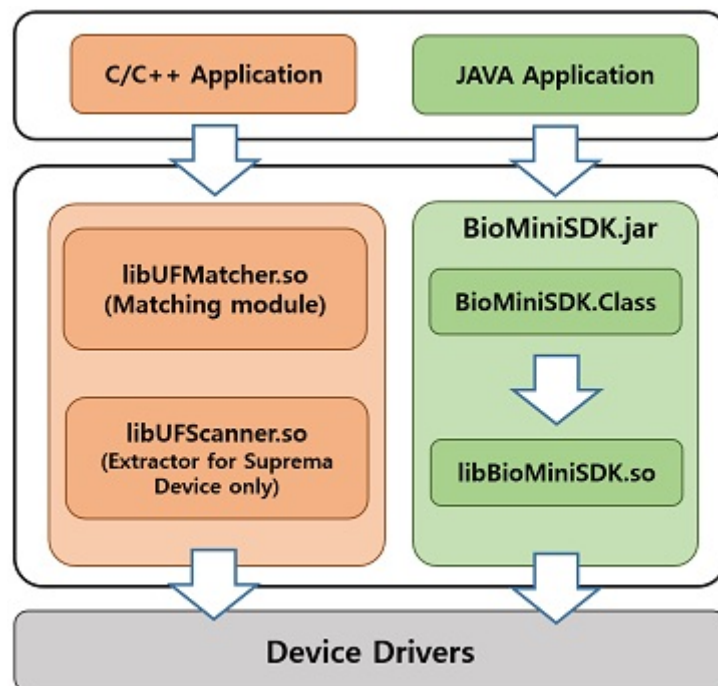
Version 3.3.0

- Support BioMini Plus scanner
- Updated extraction and matching algorithm is applied
 - The performance on the low quality fingerprints is improved
- LFD(Live Fingerprint Detection) algorithm - the live finger detection algorithm.
- Applied advanced quality measure in extraction function
- Support ISO19794-4 image format saving in BioMini SDK
- Support WSQ image format saving in BioMini SDK

Version 3.1.0

- Support BioMini scanner
- Added software module to control Suprema scanner - UFScanner
- Added software module to identify or verify a fingerprint - UFMatcher

1.3 Overview SDK Structure



BioMini SDK contains UFScanner module to control the device and UFMatcher module to perform Verify & Identification. Also, it provides Suprema.UFScanner.dll, Suprema.UFMatcher.dll for .net development and BioMiniSDK.jar interface file for JAVA development.

SDK package consists of

Path	File Name	Description
/bin	libUFMatcher.so	Matcher module
	libUFScanner.so	Scanner module
	x64\libUFScanner.so	Scanner module for x64
	x64\libUFMatcher.so	Matcher module for x64
	java\BioMiniSDK.jar	Package file for BioMiniJNI
	demoBioMiniSDK.jar	Package file for BioMini SDK java example
	demoBioMiniSDK_Java	Script file for running demoBioMiniSDK
	BioMiniSDKDemo	C++ example binary file
/docs	BioMiniSDK_for_Linux_3.6_Reference_Manual.pdf	Reference Manual
/include	UFMatcher.h	Header file for matcher module
	UFScanner.h	Header file for scanner module
/install	drivers\SFR_Linux\sfr.rules	udev rules file for BioMini Seriese device
	drivers\SFR_Linux\SFR300v2.hex	Firmware file for BioMini ver.2
	drivers\SFR_Linux\SFR400.hex	Firmware file for BioMini
	drivers\SFR_Linux\SFR500.hex	Firmware file for BioMini Plus
	HASP\Sentinel_LDK_Linux_Run-time_Installer_script.tar.gz	Sentinel HASP/LDK Linux Run-time Installer Script (You can find more information at link Link)
	HASP\install_32bit_compatibility_pacakage_for_x64.tar.gz	Sentinel HASP/LDK Script for Linux 32-bit Compatibility Packages (You can find more information at Link)
/samples	BioMiniSDKDemo\Makefile	Makefile for sample code
	BioMiniSDKDemo\BioMiniSDKDemo.cpp	Contains demo sample code for Linux(C++)
	Java\demoBioMini.java	Contains demo sample code for Java

System Requirements

The following minimum system requirements are necessary in order to use the SDK described in this document

- **Development information:**

- Linux 2.6.32, gcc 4.4.3, glibc 2.11.1 (x86 and x64)
- libusb-1.0 (essential)
- tested OS - Ubuntu(10.03), Debian(7.5), Fedora(20), CentOS(6.5), OpenSuse(13.1)

- **Supported development languages:**

- C/C++
- Java SDK 1.4 or higher (using JNI(Java Native Interface))

2. Getting Started

2.1 Driver Installation

SFR400 (BioMini)

Prerequisites

You need to install following packages.

- libusb
- udev
- fxload

Step-by-Step Installation Guide

As a root user, do the following.

1. Copy "<installed folder>/install/drivers/SFR_Linux/SFR400.hex" to /usr/share/usb. Make the directory if it does not exist in your system.

```
mkdir /usr/share/usb (if not exists)
cp SFR400.hex /usr/share/usb/
```

2. Copy "<installed folder>/install/drivers/SFR_Linux/sfr.rules" to /etc/udev directory.

```
cp sfr.rules /etc/udev
```

3. Make a symbolic link to sfr.rules as follows.
For debian 4.x

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/085-sfr.rules
```

For Fedora 6 and later, Ubuntu 6.06 and later

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/85-sfr.rules
```

4. Plug sfr400 to the system's spare usb port. If you see a red light for about a second after plugging in, the device is properly initialized and ready to use.

You may also check it by dumping usb device list and locating the product name and manufacturer string as follows:

```
cat /proc/bus/usb/devices
```

```
T: Bus=05 Lev=01 Prnt=01 Port=04 Cnt=01 Dev#= 3 Spd=480 MxCh= 0
```

```
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=16d1 ProdID=0400 Rev= 1.02
S: Manufacturer=Suprema
S: Product=SFR400
S: SerialNumber=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=00 Prot=00 Driver=(none)
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 IvL=0ms
E: Ad=86(I) Atr=02(Bulk) MxPS= 512 IvL=0ms
```

SFR500 (BioMini Plus)

Prerequisites

You need to install following packages.

- libusb
- udev
- fxload

Step-by-Step Installation Guide

As a root user, do the following.

1. Copy "<installed folder>/install/drivers/SFR_Linux/SFR500.hex" to /usr/share/usb. Make the directory if it does not exist in your system.

```
mkdir /usr/share/usb (if not exists)
cp SFR500.hex /usr/share/usb/
```

2. Copy "<installed folder>/install/drivers/SFR_Linux/sfr.rules" to /etc/udev directory.

```
cp sfr.rules /etc/udev
```

3. Make a symbolic link to sfr.rules as follows.

For debian 4.x

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/085-sfr.rules
```

For Fedora 6 and later, Ubuntu 6.06 and later

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/85-sfr.rules
```

4. Plug BioMini Plus to the system's spare usb port. If you see a light for about a second after plugging in, the device is properly initialized and ready to use.

You may also check it by dumping usb device list and locating the product name and manufacturer string as follows:

```
cat /proc/bus/usb/devices
```

```
T: Bus=05 Lev=01 Prnt=01 Port=04 Cnt=01 Dev#= 3 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=16d1 ProdID=0402 Rev= 1.02
S: Manufacturer=Suprema
S: Product=Suprema BioMini Plus
S: SerialNumber=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=00 Prot=00 Driver=(none)
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 IvL=0ms
E: Ad=86(I) Atr=02(Bulk) MxPS= 512 IvL=0ms
```

SFR410 (BioMini ver.2)

Prerequisites

You need to install following packages.

- libusb
- udev

Step-by-Step Installation Guide

As a root user, do the following.

1. Copy "<installed folder>/install/drivers/SFR_Linux/sfr.rules" to /etc/udev directory

```
cp sfr.rules /etc/udev
```

2. Make a symbolic link to sfr.rules as follows.

For debian 4.x

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/085-sfr.rules
```

For Fedora 6 and later, Ubuntu 6.06 and later

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/85-sfr.rules
```

3. Plug BioMini(ver.2) to the system's spare usb port. If you see a red light for about a second after plugging in, the device is properly initialized and ready to use.

You may also check it by dumping usb device list and locating the product name and manufacturer string as follows:

```
cat /proc/bus/usb/devices
```

```
T: Bus=05 Lev=01 Prnt=01 Port=04 Cnt=01 Dev#= 3 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=16d1 ProdID=0402 Rev= 1.02
S: Manufacturer=Suprema
S: Product=Suprema Fingerprint Scanner(Ver.02)
S: SerialNumber=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=00 Prot=00 Driver=(none)
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 IvL=0ms
E: Ad=86(I) Atr=02(Bulk) MxPS= 512 IvL=0ms
```

SFR600 (BioMini Slim)

Prerequisites

You need to install following packages.

- libusb
- udev

Step-by-Step Installation Guide

As a root user, do the following.

1. Copy "<installed folder>/install/drivers/SFR_Linux/sfr.rules" to /etc/udev directory

```
cp sfr.rules /etc/udev
```

2. Make a symbolic link to sfr.rules as follows.

For debian 4.x

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/085-sfr.rules
```

For Fedora 6 and later, Ubuntu 6.06 and later

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/85-sfr.rules
```

3. Plug BioMini Slim to the system's spare usb port. If you see a red light for about a second after plugging in, the device is properly initialized and ready to use.

You may also check it by dumping usb device list and locating the product name and manufacturer string as follows:

```
cat /proc/bus/usb/devices
```

```
T: Bus=05 Lev=01 Prnt=01 Port=04 Cnt=01 Dev#= 3 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=16d1 ProdID=0407 Rev= 1.02
S: Manufacturer=Suprema
S: Product=Suprema BioMini Slim(S20)
S: SerialNumber=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=00 Prot=00 Driver=(none)
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 IvL=0ms
E: Ad=86(I) Atr=02(Bulk) MxPS= 512 IvL=0ms
```

SFR550 (BioMini Plus2)

Prerequisites

You need to install following packages.

- libusb
- udev

Step-by-Step Installation Guide

As a root user, do the following.

1. Copy "<installed folder>/install/drivers/SFR_Linux/sfr.rules" to /etc/udev directory

```
cp sfr.rules /etc/udev
```

2. Make a symbolic link to sfr.rules as follows.

For debian 4.x

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/085-sfr.rules
```

For Fedora 6 and later, Ubuntu 6.06 and later

```
ln -s /etc/udev/sfr.rules /etc/udev/rules.d/85-sfr.rules
```

3. Plug BioMini Slim to the system's spare usb port. If you see a red light for about a second after plugging in, the device is properly initialized and ready to use.

You may also check it by dumping usb device list and locating the product name and manufacturer string as follows:

```
cat /proc/bus/usb/devices
```

```
T: Bus=01 Lev=02 Prnt=02 Port=14 Cnt=01 Dev#= 4 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=16d1 ProdID=0407 Rev= 0.00
S: Manufacturer=Suprema
S: Product=Suprema Fingerprint Scanner(Ver.02)
S: SerialNumber=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=00 Prot=00 Driver=(none)
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 IvL=0ms
E: Ad=86(I) Atr=02(Bulk) MxPS= 512 IvL=0ms
```

2.2 BioMini Demo Program

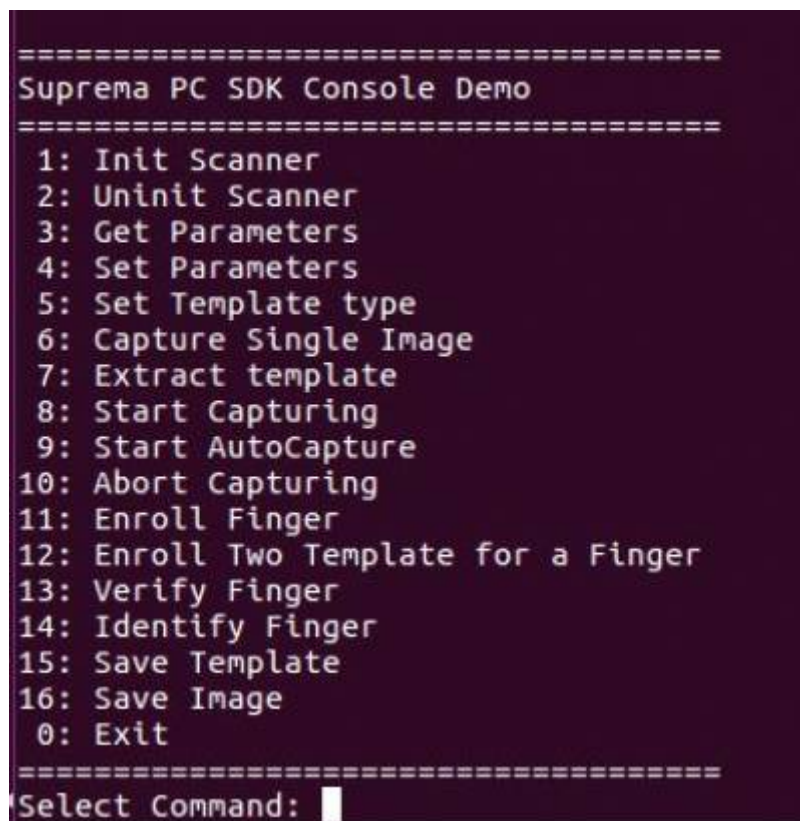
BioMini_Demo provides the basic usage about managing scanners and executing enrollment, verification and identification. This program uses UFScanner and UFMatcher modules.

Executable File Location

- bin\BioMiniSDKDemo
- bin\demoBioMini_Java
- Source code of all demo application

Picture of demo application

1. C/C++ Console demo (BioMiniSDKDemo)



```
=====
Suprema PC SDK Console Demo
=====
1: Init Scanner
2: Uninit Scanner
3: Get Parameters
4: Set Parameters
5: Set Template type
6: Capture Single Image
7: Extract template
8: Start Capturing
9: Start AutoCapture
10: Abort Capturing
11: Enroll Finger
12: Enroll Two Template for a Finger
13: Verify Finger
14: Identify Finger
15: Save Template
16: Save Image
0: Exit
=====
Select Command: █
```

2. Java demo (demoBioMini_Java)



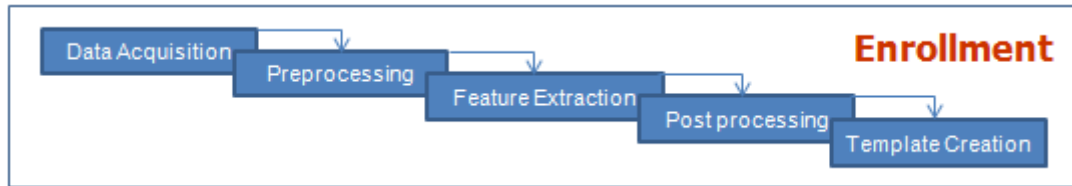
User Interface Components

BioMini SDK Demo sample demonstrates how to use BioMini SDK roughly. Please use this sample as a reference for making your own program. BioMini SDK demo provides following methods:

- Initialize Scanner - A scanner should be initialized for using all functions about the scanner
- Enrolls fingerprint - A fingerprint can be enrolled by using BioMini scanner
- Verification - A fingerprint can be verified against enrolled fingerprint
- Identification - A fingerprint can be identified against every enrolled fingerprint
- Saves data - Saves a template file or a fingerprint image to BMP format

3. C/C++ Development

3.1 Enrollment Tutorial



- **An application for people to enroll:**

- Captures multiple fingerprints for at least two fingers from a fingerprint reader.
- Checks image quality to ensure that a good quality scan is obtained.
- Extracts the fingerprint minutiae.
- Saves the fingerprint images and/or minutiae in a database.

During the enrollment process, one or more fingers are scanned for each person. We recommend that you enroll at least two fingers (more is recommended) because in the event of an accident or injury to one finger, another enrolled finger can be used to identify the individual.

The enrollment application needs to perform the following steps to enroll a single finger from a user:

Workflow

1. Call `UFS_Init()` to initialize the device for image acquisition.
2. Call `UFS_GetScannerHandle()` to get the scanner handle
3. Call `UFS_Setparameter()` to set up the parameters of the scanner
4. Call `UFS_SetTemplateType()` to set up the type of the template
5. Call `UFS_CaptureSingleImage()` to start to acquire the fingerprint image
6. Call `UFS_Uninit()` to uninitialize scanners

Example

1. Preliminaries

```
// Add Suprema UFSscanner lib (lib\UFSscanner.lib) to the Project.
// Add following statements in the source
#include "UFSscanner.h"

// We use 1024 bytes template size in this tutorial.
#define TEMPLATE_SIZE 1024
```

2. Initialize scanner module and check number of scanners

```
UFS_STATUS ufs_res;
int nScannerNumber;
```

```
// Initialize scanner module
ufs_res = UFS_Init();
// Always check status return codes after running SDK functions
// Meaning of status return code can be retrieved using UFS_GetErrorString()
// In the tutorial, we omit error check codes

// Check number of scanners
ufs_res = UFS_GetScannerNumber(&nScannerNumber);
// If number of scanner is under one, that means there is no scanner in this
system
```

3. Get first scanner

```
UFS_STATUS ufs_res;
HUFScanner hScanner;
// Get first scanner handle (0 means first scanner)
ufs_res = UFS_GetScannerHandle(, &hScanner);
```

4. Set parameters

```
// hScanner comes from section 3
UFS_STATUS ufs_res;
int nValue;
// Set timeout for capturing images to 5 seconds
nValue = 5000;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TIMEOUT, &nValue);

// Set template size to 1024 bytes
nValue = MAX_TEMPLATE_SIZE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TEMPLATE_SIZE, &nValue);

// Set not to detect core when extracting template
nValue = FALSE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_DETECT_CORE, &nValue);
```

5. Capture image and extract template

```
// hScanner comes from section 3
UFS_STATUS ufs_res;
unsigned char aTemplate[MAX_TEMPLATE_SIZE];
int nTemplateSize;
int nEnrollQuality;

// Clear capture buffer
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);
// Capture single image
```

```
ufs_res = UFS_CaptureSingleImage(hScanner);  
// If capturing images is fail, iterate above capture routine or show error  
message  
  
// Extract template from captured image  
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, aTemplate,  
&nTemplateSize, &nEnrollQuality);  
// If extraction is succeed, check nEnrollQuality is above predefined  
quality threshold
```

6. Uninitialize scanner module

```
UFS_STATUS ufs_res;  
// Uninitialize scanner module  
ufs_res = UFS_Uninit();
```

Enrollment code

```
// Add Suprema UFSscanner library (bin\Suprema.UFSscanner.dll)
// Add Suprema UFMatcher library (bin\Suprema.UFMatcher.dll) using browse
tap in the Add References dialog
UFS_STATUS ufs_res;
HUFScanner hScanner;
int nScannerNumber;

// Initialize scanner module
ufs_res = UFS_Init();
if (ufs_res == UFS_OK)
{
    // UFS_Init is succeeded
    AddMessage("UFS_Init: OK\r\n");
}
else
{
    // UFS_Init is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_Init: %s\r\n", m_strError);
    return;
}

// Check number of scanners
// If number of scanner is under one, that means there is no scanner in this
system
ufs_res = UFS_GetScannerNumber(&nScannerNumber);
if (ufs_res == UFS_OK)
{
    // UFS_GetScannerNumber is succeeded
    AddMessage("UFS_GetScannerNumner : %d\r\n", nScannerNumber);
}
else
{
    // UFS_GetScannerNumber is failed
    // UFS_UFS_GetError String function to show error srting
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_GetScannerNumner: %s\r\n", m_strError);
    return;
}

// Get first scanner handle (0 means first scanner)
ufs_res = UFS_GetScannerHandle(, &hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_GetScannerHandle is succeeded
    AddMessage("UFS_GetScannerHandle : Succeeded\r\n");
}
```

```

else
{
    // UFS_GetScannerHandle is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_GetScannerHandle: %s\r\n", m_strError);
    return;
}

int value;

// Set timeout for capturing images to 5 seconds
value = 5000;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TIMEOUT, &value);

// Set template size to 1024 bytes
value = MAX_TEMPLATE_SIZE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TEMPLATE_SIZE, &value);

// Set not to detect core when extracting template
value = FALSE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_DETECT_CORE, &value);

unsigned char Template[MAX_TEMPLATE_SIZE];
int TemplateSize;
int nEnrollQuality;

// Clear capture buffer
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_ClearCaptureImageBuffer is succeeded
    AddMessage("UFS_ClearCaptureImageBuffer : Success \r\n");
}
else
{
    // UFS_ClearCaptureImageBuffer is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ClearCaptureImageBuffer: %s\r\n", m_strError);
    return;
}

// Capture single image
ufs_res = UFS_CaptureSingleImage(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_CaptureSingleImage is succeeded
    AddMessage("UFS_CaptureSingleImage : Success \r\n");
}
else

```

```

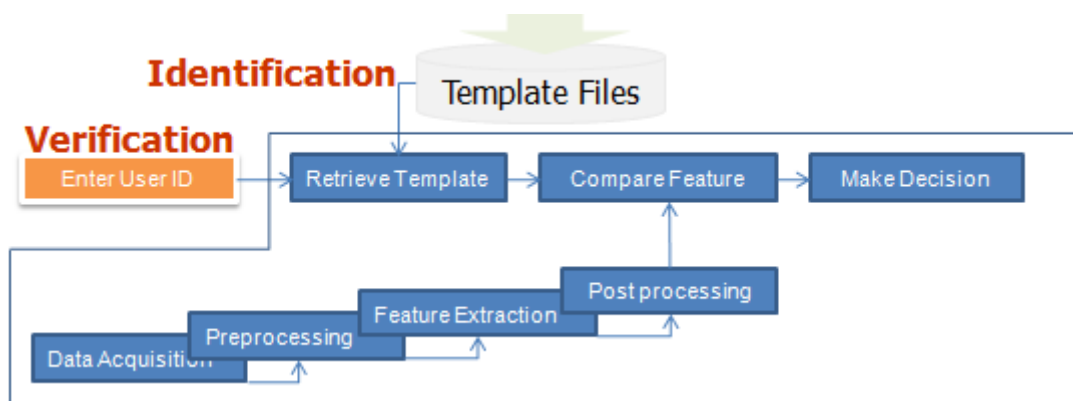
{
    //If capturing images is fail, iterate above capture routine or show
error message
    // UFS_CaptureSingleImage is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_CaptureSingleImage: %s\r\n", m_strError);
    return;
}

// Extract template from captured image
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, Template,
&TemplateSize, &nEnrollQuality);
if (ufs_res == UFS_OK)
{
    // If extraction is succeed, check nEnrollQuality is above predefined
quality threshold
    // UFS_ExtractEx is succeeded
    AddMessage("UFS_ExtractEx : Success \r\n");
}
else
{
    // UFS_ExtractEx is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ExtractEx: %s\r\n", m_strError);
}

// Uninitialize scanner module
ufs_res = UFS_Uninit();
if (ufs_res == UFS_OK)
{
    // UFS_Uninit is succeeded
    AddMessage("UFS_Uninit : Success \r\n");
}
else
{
    // UFS_Uninit is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_Uninit: %s\r\n", m_strError);
}
}

```

3.2 Verification Tutorial



- **Fingerprint recognition involves operation:**

- **Verifying** - Comparing a fingerprint against a specific user's enrolled fingerprint(s) to verify a specific person's identity (e.g., when the user types their name and then uses a fingerprint rather than a password).

Verification Workflow

1. `UFS_Init()` to initialize scanners for image acquisition
2. `UFS_GetScannerHandle()` to get the scanner handle
3. `UFS_Setparameter()` to set up the parameters of the scanner
4. `UFS_SetTemplateType()` to set up the type of the template
5. `UFS_CaptureSingleImage()` to start to acquire the fingerprint image
6. `UFS_Extract()` to extract the captured image to template
7. `UFM_Create()` to create a Matcher for matching
8. `UFM_Verify()` to compare it to a selected template in database
9. `UFM_Delete()` to close a Matcher
10. `UFS_Uninit()` to uninitialize scanners

Example

1. Preliminaries

```
// Add Suprema UFMatcher lib (lib\UFMatcher.lib) to the Project
// Add following statements in the source
#include "UFMatcher.h"

// We use 1024 bytes template size in this tutorial
#define MAX_TEMPLATE_SIZE 1024
```

2. Initialize scanner module and check number of scanners

```
UFM_STATUS ufm_res; HUFMatcher hMatcher;
// Create matcher
ufm_res = UFM_Create(&hMatcher);
```



```
// Always check status return codes after running SDK functions
// Meaning of status return code can be retrieved using UFM_GetErrorString()
// In the tutorial, we omit error check codes
```

3. Get first scanner

```
UFS_STATUS ufs_res;
HUFSscanner hScanner;

// Get first scanner handle (0 means first scanner)
ufs_res = UFS_GetScannerHandle(, &hScanner);
```

4. Set parameters

```
// hScanner comes from section 3
UFS_STATUS ufs_res;
int value;

// Set timeout for capturing images to 5 seconds
value = 5000;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TIMEOUT, &value);

// Set template size to 1024 bytes
value = MAX_TEMPLATE_SIZE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TEMPLATE_SIZE, &value);

// Set not to detect core when extracting template
value = FALSE;
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_DETECT_CORE, &value);
```

5. Capture image and extract template

```
// hScanner comes from section 3
UFS_STATUS ufs_res;
byte[] Template = new byte[MAX_TEMPLATE_SIZE];
int TemplateSize;
int EnrollQuality;

// Clear capture buffer
ufs_res = Scanner.ClearCaptureImageBuffer();

// Capture single image
ufs_res = Scanner.CaptureSingleImage();
// If capturing images is fail, iterate above capture routine or show error
message

// Extract template from captured image
```

```
ufs_res = Scanner.ExtractEx(MAX_TEMPLATE_SIZE, Template, out TemplateSize,  
out EnrollQuality);  
// If extraction is succeed, check nEnrollQuality is above predefined  
quality threshold
```

6. Uninitialize scanner module

```
UFS_STATUS ufs_res;  
  
// Uninitialize scanner module  
ufs_res = UFS_Uninit();
```

verification code

```
// Add Suprema UFSscanner library (bin\Suprema.UFSscanner.dll)
// Add Suprema UFMatcher library (bin\Suprema.UFMatcher.dll) using browse
tap in the Add References dialog
UFS_STATUS ufs_res;
HUFScanner hScanner;

UFM_STATUS ufm_res;
HUFMatcher hMatcher;
int nScannerNumber;

// Initialize scanner module
ufs_res = UFS_Init();
if (ufs_res == UFS_OK)
{
    // UFS_Init is succeeded
    AddMessage("UFS_Init: OK\r\n");
}
else
{
    // UFS_Init is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_Init: %s\r\n", m_strError);
    return;
}

// Get first scanner handle (0 means first scanner)
ufs_res = UFS_GetScannerHandle(, &hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_GetScannerHandle is succeeded
    AddMessage("UFS_GetScannerHandle : Succeeded\r\n");
}
else
{
    // UFS_GetScannerHandle is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_GetScannerHandle: %s\r\n", m_strError);
    return;
}

// Create matcher
ufm_res = UFM_Create(&hMatcher);
// Always check status return codes after running SDK functions
// Meaning of status return code can be retrieved using UFM_GetErrorString()
// In the tutorial, we omit error check codes
if (ufm_res == UFM_OK)
{

```

```

    // UFM_Create is succeeded
    AddMessage("UFM_Create : Success \r\n");
}
else
{
    // UFM_Create is failed
    // Use UFM_GetErrorString function to show error string
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_Create: %s\r\n", m_strError);
}

int value;
// Set security level to 3
value = 3;
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_SECURITY_LEVEL, &value);
if (ufm_res == UFM_OK)
{
    AddMessage("UFM_SetParameter : Success \r\n");
}
else
{
    //Error handling routine is omitted
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_SetParameter: %s\r\n", m_strError);
}

unsigned char Template1[MAX_TEMPLATE_SIZE];
int TemplateSize1;
int nEnrollQuality;

// Clear capture buffer
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_ClearCaptureImageBuffer is succeeded
    AddMessage("UFS_ClearCaptureImageBuffer : Success \r\n");
}
else
{
    // UFS_ClearCaptureImageBuffer is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ClearCaptureImageBuffer: %s\r\n", m_strError);
    return;
}

// Capture single image
ufs_res = UFS_CaptureSingleImage(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_CaptureSingleImage is succeeded

```

```

    AddMessage("UFS_CaptureSingleImage : Success \r\n");
}
else
{
    //If capturing images is fail, iterate above capture routine or show
error message
    // UFS_CaptureSingleImage is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_CaptureSingleImage: %s\r\n", m_strError);
    return;
}

// Extract template from captured image
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, Template1,
&TemplateSize1, &nEnrollQuality);
if (ufs_res == UFS_OK)
{
    // If extraction is succeed, check nEnrollQuality is above predefined
quality threshold
    // UFS_ExtractEx is succeeded
    AddMessage("UFS_ExtractEx : Success \r\n");
}
else
{
    // UFS_ExtractEx is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ExtractEx: %s\r\n", m_strError);
}

unsigned char Template2[MAX_TEMPLATE_SIZE];
int TemplateSize2;
int nEnrollQuality;

// Clear capture buffer
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_ClearCaptureImageBuffer is succeeded
    AddMessage("UFS_ClearCaptureImageBuffer : Success \r\n");
}
else
{
    // UFS_ClearCaptureImageBuffer is failed
    // Use UFS_GetErrorString function to show error srting
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ClearCaptureImageBuffer: %s\r\n", m_strError);
    return;
}

```

```

// Capture single image
ufs_res = UFS_CaptureSingleImage(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_CaptureSingleImage is succeeded
    AddMessage("UFS_CaptureSingleImage : Success \r\n");
}
else
{
    //If capturing images is fail, iterate above capture routine or show
error message
    // UFS_CaptureSingleImage is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_CaptureSingleImage: %s\r\n", m_strError);
    return;
}

// Extract template from captured image
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, Template2,
&TemplateSize2, &nEnrollQuality);
if (ufs_res == UFS_OK)
{
    // If extraction is succeed, check nEnrollQuality is above predefined
quality threshold
    // UFS_ExtractEx is succeeded
    AddMessage("UFS_ExtractEx : Success \r\n");
}
else
{
    // UFS_ExtractEx is failed
    // Use UFS_GetErrorString function to show error string
    UFM_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ExtractEx: %s\r\n", m_strError);
}

int bVerifySucceed;
// Verify two templates
ufm_res = UFM_Verify(hMatcher, Template1, TemplateSize1, Template2,
TemplateSize2, &bVerifySucceed);
if (ufm_res == UFM_OK)
{
    // UFM_Verify is succeeded
    if (bVerifySucceed)
    {
        // Template1 is matched to Template2
        AddMessage("Template1 is matched to Template2 \r\n");
    }
    else
    {
        // Template1 is not matched to Template2
    }
}

```

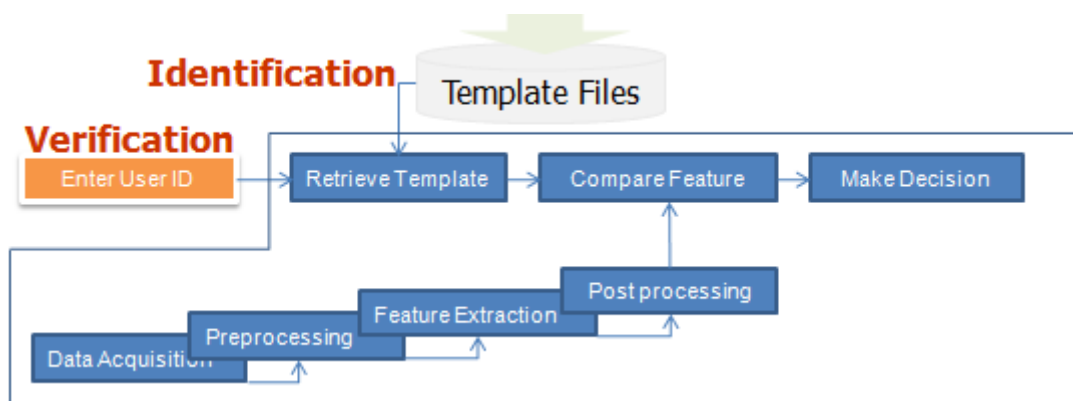
```

        AddMessage("Template1 is not matched to Template2 \r\n");
    }
}
else
{
    // UFM_Verify is failed
    UFM_GetErrorString(ufs_res, m_strError);
    AddMessage("UFM_Verify: %s\r\n", m_strError);
}

// Delete matcher
ufm_res = UFM_Delete(&hMatcher);
if (ufm_res == UFM_OK)
{
    // UFM_Delete is succeeded
    AddMessage("UFM_Delete : Success \r\n");
}
else
{
    // UFM_Delete is failed
    // Use UFM_GetErrorString function to show error string
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_Delete: %s\r\n", m_strError);
}
}

```

3.3 Identification Tutorial



- **Fingerprint recognition involves operation:**

- **Identification** - Comparing a fingerprint against the database of enrolled fingerprints and confirming that the fingerprint is enrolled (e.g., to open a door there may be many authorized users).

Identification Workflow

1. `UFS_Init()` to initialize scanners for image acquisition
2. `UFS_GetScannerHandle()` to get the scanner handle
3. `UFS_Setparameter()` to set up the parameters of the scanner
4. `UFS_SetTemplateType()` to set up the template type
5. `UFS_CaptureSingleImage()` to start to acquire the fingerprint image
6. `UFS_Extract()` to extract the captured image to template
7. `UFM_Create()` to create a Matcher for matching
8. `UFM_Identify()` to compare it to N number of templates in database
9. `UFM_Delete()` to close a Matcher
10. `UFS_Uninit()` to uninitialize scanners

Example

1. Preliminaries

```
// Add Suprema UFMatcher lib (lib\UFMatcher.lib) to the Project
// Add following statements in the source
#include "UFMatcher.h"

// We use 1024 bytes template size in this tutorial
#define MAX_TEMPLATE_SIZE 1024
// Set maximum template number to 50 (number depends on application)
#define MAX_TEMPLATE_NUM 50
```

2. Create matcher

```
UFM_STATUS ufm_res;
```



```
HUFMatcher hMatcher;

// Create matcher
ufm_res = UFM_Create(&hMatcher);

// Always check status return codes after running SDK functions
// Meaning of status return code can be retrieved using UFM_GetErrorString()
// In the tutorial, we omit error check codes
```

3. Set parameters

```
// hMatcher comes from section 2
UFM_STATUS ufm_res;
int value;

// Set security level to 4
value = 4;
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_SECURITY_LEVEL, &value);

// Set fast mode on
value = TRUE;
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_FAST_MODE, &value);
```

4. Identify

```
// hMatcher comes from section 2
UFM_STATUS ufm_res;
unsigned char Template1[MAX_TEMPLATE_SIZE];
int Template1Size;
unsigned char* Template2Array[MAX_TEMPLATE_NUM];
int Template2SizeArray[MAX_TEMPLATE_NUM];
int Template2Num;
int nMatchIndex;

// Allocate Template2Array
for (i = 0; i < MAX_TEMPLATE_NUM; i++)
{
    Template2Array[i] = (unsigned
    char*)malloc(MAX_TEMPLATE_SIZE);
    memset(Template2Array[i], 0, MAX_TEMPLATE_SIZE);
}

// Get Template1 from scanner or image or database
// Get Template2Array from scanner or image or database

// Identify Template1 from Template2Array, set timeout to 5
seconds
ufm_res = UFM_Identify(hMatcher, Template1, Template1Size, Template2Array,
Template2SizeArray, Template2Num, 5000, &nMatchIndex);
```

```

if (ufm_res != UFM_OK)
{
    // Execute error handling codes
}
else
{
    if (nMatchIndex != -1)
    {
        // Identification succeed
    }
    else
    {
        // Identification failed
    }
}

// Free Template2Array
for (i = ; i < MAX_TEMPLATE_NUM; i++)
{
    free(Template2Array[i]);
}

```

5. Delete matcher

```

// hMatcher comes from section 2
UFM_STATUS ufm_res;

// Delete matcher
ufm_res = UFM_Delete(&hMatcher);

```

Identification code

```
// Add Suprema UFSscanner library (bin\Suprema.UFSscanner.dll)
// Add Suprema UFMatcher library (bin\Suprema.UFMatcher.dll) using browse
tap in the Add References dialog

UFS_STATUS ufs_res;
HUFSscanner hScanner;

UFM_STATUS ufm_res;
HUFMatcher hMatcher;
int nScannerNumber;

// Initialize scanner module
ufs_res = UFS_Init();
if (ufs_res == UFS_OK)
{
    // UFS_Init is succeeded
    AddMessage("UFS_Init: OK\r\n");
}
else
{
    // UFS_Init is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_Init: %s\r\n", m_strError);
    return;
}

// Get first scanner handle (0 means first scanner)
ufs_res = UFS_GetScannerHandle(, &hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_GetScannerHandle is succeeded
    AddMessage("UFS_GetScannerHandle : Succeeded\r\n");
}
else
{
    // UFS_GetScannerHandle is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_GetScannerHandle: %s\r\n", m_strError);
    return;
}

// Create matcher
ufm_res = UFM_Create(&hMatcher);
// Always check status return codes after running SDK functions
// Meaning of status return code can be retrieved using UFM_GetErrorString()
// In the tutorial, we omit error check codes
if (ufm_res == UFM_OK)
```

```

{
    // UFM_Create is succeeded
    AddMessage("UFM_Create : Success \r\n");
}
else
{
    // UFM_Create is failed
    // Use UFM_GetErrorString function to show error string
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_Create: %s\r\n", m_strError);
}

int value;
// Set security level to 4
value = 4;

ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_SECURITY_LEVEL, &value);
if (ufm_res == UFM_OK)
{
    AddMessage("UFM_SetParameter : Success \r\n");
}
else
{
    //Error handling routine is omitted
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_SetParameter: %s\r\n", m_strError);
}

// Set fast mode on
value = TRUE;

ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_FAST_MODE, &value);
if (ufm_res == UFM_OK)
{
    AddMessage("UFM_SetParameter : Success \r\n");
}
else
{
    //Error handling routine is omitted
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_SetParameter: %s\r\n", m_strError);
}

unsigned char Template1[MAX_TEMPLATE_SIZE];
unsigned char** ppTemplate2;
int nTemplate1Size;
int* pnTemplate2Size;
int nTemplate2Num;
int nTimeout;
int nMatchTemplate2Index;
int i;

```

```

int nEnrollQuality;

// Get input template from user, Template1
// Clear capture buffer
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_ClearCaptureImageBuffer is succeeded
    AddMessage("UFS_ClearCaptureImageBuffer : Success \r\n");
}
else
{
    // UFS_ClearCaptureImageBuffer is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ClearCaptureImageBuffer: %s\r\n", m_strError);
    return;
}

// Capture single image
ufs_res = UFS_CaptureSingleImage(hScanner);
if (ufs_res == UFS_OK)
{
    // UFS_CaptureSingleImage is succeeded
    AddMessage("UFS_CaptureSingleImage : Success \r\n");
}
else
{
    //If capturing images is fail, iterate above capture routine or show
error message
    // UFS_CaptureSingleImage is failed
    // Use UFS_GetErrorString function to show error string
    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_CaptureSingleImage: %s\r\n", m_strError);
    return;
}

// Extract template from captured image
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, Template1,
&nTemplate1Size, &nEnrollQuality);
if (ufs_res == UFS_OK)
{
    // If extraction is succeed, check nEnrollQuality is above predefined
quality threshold
    // UFS_ExtractEx is succeeded
    AddMessage("UFS_ExtractEx : Success \r\n");
}
else
{
    // UFS_ExtractEx is failed
    // Use UFS_GetErrorString function to show error string

```

```

    UFS_GetErrorString(ufs_res, m_strError);
    AddMessage("UFS_ExtractEx: %s\r\n", m_strError);
}

// Make template array from DB or something
// Get number of template to nTemplate2Num

ppTemplate2 = (unsigned char**)malloc(nTemplate2Num * sizeof(unsigned
char*));
pnTemplate2Size = (int*)malloc(nTemplate2Num * sizeof(int));
for (i = ; i < nTemplate2Num; i++)
{
    ppTemplate2[i] = (unsigned char*)malloc(MAX_TEMPLATE_SIZE *
sizeof(unsigned char));
    // Copy i th template to ppTemplate2[i]
    // Set i th template size to pnTemplateSize[i]
}

// Set match timeout to nTimeout

ufm_res = UFM_Identify(hMatcher, Template1, nTemplate1Size, ppTemplate2,
pnTemplate2Size, nTemplate2Num, nTimeout, &nMatchTemplate2Index);
if (ufm_res == UFM_OK)
{
    // UFM_Identify is succeed
    AddMessage("UFM_Identify : Success \r\n");
    if (nMatchTemplate2Index != -1)
    {
        // Input fingerprint Template1 is matched to
ppTemplate2[nMatchTemplate2Index]
        AddMessage("Matching Succeeded : [%d]", &nMatchTemplate2Index);
    }
    else
    {
        // Input fingerprint is not in ppTemplate2
        AddMessage("Matching Failed");
    }
}
else
{
    // UFM_Identify is failed
    // Use UFM_GetErrorString function to show error string
    UFM_GetErrorString(ufm_res, m_strError);
    AddMessage("UFM_Identify: %s\r\n", m_strError);
}

// Delete matcher
ufm_res = UFM_Delete(&hMatcher);
if (ufm_res == UFM_OK)
{
    // UFM_Delete is succeeded

```

```
    AddMessage("UFM_Delete : Success \r\n");
}
else
{
    // UFM_Delete is failed
    // Use UFM_GetErrorString function to show error string
    UFM_GetErrorString(uvm_res, m_strError);
    AddMessage("UFM_Delete: %s\r\n", m_strError);
}
```

3.4 C/C++ APIs



Below are instructions of the C++ APIs. You can find the detailed information of the C++ APIs in a header file. This defines the class and structure of the data that are necessary to use the functions. All header files are located in <SDK_HOME>\include folder.

UFS_Init()

Initializes a UFScanner module

UFS_Update()

Enforces a UFScanner module to update the connection state of scanners

UFS_Uninit()

Un-initializes a UFScanner module

UFS_SetScannerCallback()

Registers a scanner callback function

UFS_RemoveScannerCallback()

Removes a registered scanner callback function

UFS_GetScannerNumber()

Gets the number of scanners

UFS_GetScannerHandle()

Gets the scanner handle using the scanner index

UFS_GetScannerHandleByID()

Gets the scanner handle using a scanner ID

UFS_GetScannerIndex()

Gets the scanner index that is assigned to the scanner handle

UFS_GetScannerID()

Gets a scanner ID that is assigned to the scanner handle

UFS_GetScannerType()

Gets the scanner type that is assigned to the scanner handle

UFS_GetParameter()

Gets the parameter value of a UFScanner module

UFS_SetParameter()

Sets the parameter value of a UFScanner module

UFS_IsSensorOn()

Checks whether a scanner is connected or not

UFS_IsFingerOn()

Checks whether a finger is placed on a scanner or not

UFS_CaptureSingleImage()

Captures single image. Captured image is stored to the internal buffer

UFS_StartCapturing()

Starts capturing. The capture is going on until the specified time exceeds

UFS_StartAutoCapture()

Starts the automatic capture. Currently this function is working for Suprema SFR600(BioMini Slim) only

UFS_IsCapturing()

Checks if the specified scanner is running to capture which is started by UFS_CaptureSingleImage or UFS_StartCapturing

UFS_AbortCapturing()

Aborts capturing which is started by UFS_CaptureSingleImage or UFS_StartCapturing

UFS_Extract()

Extracts a template from the stored image buffer which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing()

UFS_ExtractEx()

Extracts a template from the stored image buffer which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing(). This is extended version of UFS_Extract function to accommodate a template with large size

UFS_SetEncryptionKey()

Sets the encryption key

UFS_EncryptTemplate()

Encrypts a template

UFS_DecryptTemplate()

Decrypts a template

UFS_GetCaptureImageBufferInfo()

Gets the information of the capture image buffer

UFS_GetCaptureImageBuffer()

Copies the capture image buffer to the specified image data array

UFS_GetCaptureImageBufferToBMPIImageBuffer()

Copies the capture image buffer to the specified image data of bmp format

UFS_GetCaptureImageBufferTo19794_4ImageBuffer()

Copies the capture image buffer to the specified image data of 19794_4 format

[UFS_GetCaptureImageBufferToWSQImageBuffer\(\)](#)

Copies the capture image buffer to the specified image data of the WSQ format

[UFS_GetCaptureImageBufferToWSQImageBufferVar\(\)](#)

Copies the capture image buffer (cropped or expanded to the specified size) to the target image data buffer of the WSQ format

[UFS-DecompressWSQBMP\(\)](#)

Decompress a WSQ file and save to a BMP file

[UFS-DecompressWSQBMPMem\(\)](#)

Decompress a WSQ buffer and save to the image data of the bmp format

[UFS_DrawCaptureImageBuffer\(\)](#)

Draws the fingerprint image which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing(). This function is not supported on java

[UFS_DrawFeatureBuffer\(\)](#)

Draws the fingerprint image which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing(). This function is not supported on java And should be called after the extraction from the last captured fingerprint image. If extraction is not performed from the last captured image, this function will not draw the feature in the image frame

[UFS_SaveCaptureImageBufferToBMP\(\)](#)

Saves the capture image buffer to the specified file of the bitmap format

[UFS_SaveCaptureImageBufferTo19794_4\(\)](#)

Saves the capture image buffer to the specified file of the 19794_4 format

[UFS_SaveCaptureImageBufferToWSQ\(\)](#)

Saves the capture image buffer to the specified file of the WSQ format

[UFS_SaveCaptureImageBufferToWSQVar\(\)](#)

Saves the capture image buffer (cropped or expanded to the specified size) to the target file of the WSQ format

[UFS_ClearCaptureImageBuffer\(\)](#)

Clears the capture image buffer stored to the internal buffer

[UFS_GetErrorString\(\)](#)

Gets the error string for specified UFS_STAUS value

[UFS_GetTemplateType\(\)](#)

Gets the template type value

[UFS_SetTemplateType\(\)](#)

Sets the template type value

[UFS_SelectTemplate\(\)](#)

Selects n number of good templates from m number of input templates

UFS_SelectTemplateEx()

Selects n number of good templates from m number of input templates This is extended version of UFS_SelectTemplate function to accommodate the template with large size

UFS_GetFPQuality()

Calculates the quality score of an image as defined in NISTIR 7151: FingerPrint Image Quality. The score would be between 1(excellent) and 5(poor)

UFS_GetFeatureNumber()

Get number of Minutiae from template data

UFS_EnrollUI()

Generates the fingerprint enrollment dialog. This function can be called after executing UFS_Init(). Enrolling a fingerprint is extracting a template from finger and saving the template. Below sample's UFS_EnrollUI() captures a fingerprint image after setting the template type. And extracts a template from captured fingerprint image. The extracted template will be saved in a specific template array, which is a parameter of the UFS_EnrollUI(). It supported only for Windows environment

UFS_VerifyUI()

Generates the fingerprint verification dialog. This function can be called after executing UFS_Init() and UFS_EnrollUI(). Two fingerprints can be verified whether they are matched or not. Below sample's UFS_VerifyUI() captures a fingerprint image and extracts a template from the image. And execute 1:1 matching using extracted template and templates enrolled from UFS_EnrollUI()

UFM_Create()

Creates a matcher object

UFM_Delete()

Deletes a specified matcher object

UFM_GetParameter()

Gets the parameter value of UFMatcher module

UFM_SetParameter()

Sets the parameter value of UFMatcher module

UFM_Verify()

Compares two extracted templates

UFM_Identify(),IdentifyMT()

Compares a template with given template array UFM_IdentifyMT function uses multi threads internally for faster identifying in multi-core systems

UFM_AbortIdentify()

Aborts current identifying procedure started using UFM_Identify()

UFM_IdentifyInit()

Initializes identify with input template

UFM_IdentifyNext()

Matches one input template to the template specified in UFM_IdentifyInit()

UFM_RotateTemplate()

Rotates the specified template to the amount of 180 degrees

UFM_GetErrorString()

Gets the error string for the specified UFM_STAUS value

UFM_GetTemplateType()

Gets the parameter value

UFM_SetTemplateType()

Sets the parameter value

UFS_Init

Initializes a UFSscanner module.

```
UFS_STATUS UFS_API UFS_Init();
```

Examples

```
UFS_STATUS ufs_res;  
int nScannerNumber;  
  
ufs_res = UFS_GetScannerNumber(&nScannerNumber);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerNumber is succeeded  
}  
else  
{  
    // UFS_GetScannerNumber is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_Update

Enforces a UFSscanner module to update the connection state of scanners.

```
UFS_STATUS UFS_API UFS_Update();
```

Examples

```
UFS_STATUS ufs_res;  
ufs_res = UFS_Update();  
if (ufs_res == UFS_OK)  
{  
    // UFS_Init is succeeded  
}  
else  
{  
    // UFS_Init is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_Uninit

Un-initializes a UFSscanner module.

```
UFS_STATUS UFS_API UFS_Uninit();
```

Examples

```
UFS_STATUS ufs_res;

ufs_res = UFS_Uninit();
if (ufs_res == UFS_OK)
{
    // UFS_Init is succeeded
}
else
{
    // UFS_Init is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values([refer to return values](#))

UFS_SetScannerCallback

Registers the scanner callback function.

```
UFS_STATUS UFS_API UFS_SetScannerCallback(  
    UFS_SCANNER_PROC* pScannerProc,  
    void* pParam );
```

Parameters

- *pScannerProc [in]* : Handle to the UFS_SCANNER_PROC function which receives scanner events
- *pParam [in]* : Pointer to the scanner callback data which will be transmitted with a scanner callback event

Examples

```
UFS_STATUS ufs_res;  
  
ufs_res = UFS_Uninit();  
if (ufs_res == UFS_OK)  
{  
    // UFS_Init is succeeded  
}  
else  
{  
    // UFS_Init is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_RemoveScannerCallback

Removes the registered scanner callback function.

```
UFS_STATUS UFS_API UFS_RemoveScannerCallback();
```

Examples

```
UFS_STATUS ufs_res;

ufs_res = UFS_RemoveScannerCallback();
if (ufs_res == UFS_OK)
{
    // UFS_RemoveScannerCallback is succeeded
}
else
{
    // UFS_RemoveScannerCallback is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values([refer to return values](#))

UFS_GetScannerNumber

Gets the number of scanners.

```
UFS_STATUS UFS_API UFS_GetScannerNumber(  
    int* pnScannerNumber);
```

Parameters

- *pnScannerNumber [out]* : Receive the number of scanners

Examples

```
UFS_STATUS ufs_res;  
int nScannerNumber;  
  
ufs_res = UFS_GetScannerNumber(&nScannerNumber);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerNumber is succeeded  
}  
else  
{  
    // UFS_GetScannerNumber is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerHandle

Gets the scanner handle using a scanner index.

```
UFS_STATUS UFS_API UFS_GetScannerHandle(  
    int nScannerIndex,  
    HUFSScanner* phScanner );
```

Parameters

- *nScannerIndex [in]* : Scanner index (0 ~ number of scanners - 1)
- *phScanner [out]* : Pointer to handle of the scanner object

Examples

```
UFS_STATUS ufs_res;  
int nScannerIndex;  
HUFSScanner hScanner;  
  
// Set nScannerIndex to (0 ~ number of scanners - 1 )  
// Number of scanner can be retrieved using UFS_GetScannerNumber function  
  
ufs_res = UFS_GetScannerHandle(nScannerIndex, &hScanner);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerHandle is succeeded  
}  
else  
{  
    // UFS_GetScannerHandle is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerHandleByID

Gets the scanner handle using a scanner ID.

```
UFS_STATUS UFS_API UFS_GetScannerHandleByID(  
    const char* szScannerID,  
    HUFScanner* phScanner );
```

Parameters

- *szScannerID [in]* : Scanner ID
- *phScanner [out]* : Pointer to handle of the scanner object

Examples

```
UFS_STATUS ufs_res;  
char strID[64];  
HUFScanner hScanner;  
// Assign scanner ID to strID  
// Scanner ID can be retrieved using UFS_GetScannerID function  
  
ufs_res = UFS_GetScannerHandleByID(strID, &hScanner);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerHandleByID is succeeded  
}  
else  
{  
    // UFS_GetScannerHandleByID is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerIndex

Gets a scanner index that is assigned to the scanner handle.

```
UFS_STATUS UFS_API UFS_GetScannerIndex(  
    HUFScanner hScanner,  
    int* pnScannerIndex );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnScannerIndex [out]* : Receive scanner index of specified scanner handle

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nScannerIndex;  
// Get hScanner handle  
  
ufs_res = UFS_GetScannerIndex(hScanner, &nScannerIndex);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerIndex is succeeded  
}  
else  
{  
    // UFS_GetScannerIndex is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerID

Gets scanner ID assigned to scanner handle.

```
UFS_STATUS UFS_API UFS_GetScannerID(  
    HUFScanner hScanner,  
    char* szScannerID );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szScannerID [out]* : Receive scanner ID of specified scanner handle; Scanner ID has maximum 32 characters. szScannerID must be allocated in user's applications and allocated size must be larger than 33 bytes for considering null character in 33th byte position.

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
char strID[64];  
// Should be larger than 33 bytes  
// Get hScanner handle  
  
ufs_res = UFS_GetScannerID(hScanner, strID);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerID is succeeded  
}  
else  
{  
    // UFS_GetScannerID is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerType

Gets the scanner type that is assigned to the scanner handle.

```
UFS_STATUS UFS_API UFS_GetScannerType(  
    HUFSscanner hScanner,  
    int* pnScannerType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnScannerType [out]* : Receives one of the scanner type

Scanner type	Code	Description
UFS_SCANNER_TYPE_SFR200	1001	Suprema SFR200
UFS_SCANNER_TYPE_SFR300	1002	Suprema SFR300-S
UFS_SCANNER_TYPE_SFR300v2	1003	Suprema SFR300v2, SFR400
UFS_SCANNER_TYPE_SFR500	1004	Suprema SFR500
UFS_SCANNER_TYPE_SFR600	1005	Suprema SFR600
UFS_SCANNER_TYPE_SFR410	1006	Suprema SFR410
UFS_SCANNER_TYPE_SFR550	1007	Suprema SFR550

Examples

```
UFS_STATUS ufs_res;  
HUFSscanner hScanner;  
int nScannerType;  
// Get hScanner handle  
  
ufs_res = UFS_GetScannerType(hScanner, &nScannerType);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerType is succeeded  
}  
else  
{  
    // UFS_GetScannerType is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_GetParameter

Gets parameter value of UFSscanner module.

```
UFS_STATUS UFS_API UFS_GetParameter(  
    HUFScanner hScanner,  
    int nParam,  
    void* pValue );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFS_PARAM_TIMEOUT	201	Timeout (millisecond unit) (0: infinite)	5000
UFS_PARAM_BRIGHTNESS	202	Brightness (0 ~ 255); Higher value means darker image. * Not supported Device: BioMini-Slim(SFU-S20, SFU-S20B) BioMini(SFU-300)	100
UFS_PARAM_SENSITIVITY	203	Sensitivity (0 ~ 7); Higher value means more sensitive	4
UFS_PARAM_SERIAL	204	Serial (get only)	-
UFS_PARAM_SDK_VERSION	210	SDK Version (get only)	
UFS_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	
UFS_PARAM_DETECT_CORE	301	Detect core (0: not use core, 1: use core)	0
UFS_PARAM_TEMPLATE_SIZE	302	Template size (byte unit) (256 ~ 1024, 32 bytes step size)	1024
UFS_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFS_PARAM_DETECT_FAKE	312	Use live Finger Detection (0: not use LFD, 1 ~ 3 : use LFD); Higher value means more strong to fake finger * Supported Device: BioMini Slim(SFU-S20, SFU-S20B)	0

- *pValue [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
int nValue;  
char strSerial[64];  
char strSdkVer[64];  
char copyright[64];  
  
// Get hScanner handle  
  
// Get timeout  
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_TIMEOUT, &nValue);  
// Error handling routine is omitted
```



```

// Get brightness
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_BRIGHTNESS, &nValue);
// Error handling routine is omitted

// Get sensitivity
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_SENSITIVITY, &nValue);
// Error handling routine is omitted

// Get serial
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_SERIAL, strSerial);
// Error handling routine is omitted

// Get SDK version
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_SDK_VERSION, strSdkVer);
// Error handling routine is omitted

// Get SDK copyright
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_SDK_COPYRIGHT, copyright);
// Error handling routine is omitted

// Get detect core
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_DETECT_CORE, &nValue);
// Error handling routine is omitted

// Get template size
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_TEMPLATE_SIZE, &nValue);
// Error handling routine is omitted

// Get use SIF
ufs_res = UFS_GetParameter(hScanner, UFS_PARAM_USE_SIF, &nValue);
// Error handling routine is omitted

```

Return Values([refer to return values](#))

UFS_SetParameter

Sets parameter value of UFSscanner module.

```
UFS_STATUS UFS_API UFS_SetParameter(  
    HUFScanner hScanner,  
    int nParam,  
    void* pValue );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFS_PARAM_TIMEOUT	201	Timeout (millisecond unit) (0: infinite)	5000
UFS_PARAM_BRIGHTNESS	202	Brightness (0 ~ 255); Higher value means darker image. * Not supported Device: BioMini-Slim(SFU-S20, SFU-S20B) BioMini(SFU-300)	100
UFS_PARAM_SENSITIVITY	203	Sensitivity (0 ~ 7); Higher value means more sensitive	4
UFS_PARAM_SERIAL	204	Serial (get only)	-
UFS_PARAM_SDK_VERSION	210	SDK Version (get only)	
UFS_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	
UFS_PARAM_DETECT_CORE	301	Detect core (0: not use core, 1: use core)	0
UFS_PARAM_TEMPLATE_SIZE	302	Template size (byte unit) (256 ~ 1024, 32 bytes step size)	1024
UFS_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFS_PARAM_DETECT_FAKE	312	Use live Finger Detection (0: not use LFD, 1 ~ 3 : use LFD); Higher value means more strong to fake finger * Supported Device: BioMini Slim(SFU-S20, SFU-S20B)	0

- *pValue [in]* : Pointer to parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
int nValue;  
  
// Get hScanner handle  
  
// Set timeout to nValue  
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TIMEOUT, &nValue);  
// Error handling routine is omitted  
  
// Set brightness to nValue  
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_BRIGHTNESS, &nValue);  
// Error handling routine is omitted
```

```
// Set sensitivity to nValue
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_SENSITIVITY, &nValue);
// Error handling routine is omitted

// Set detect core to nValue
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_DETECT_CORE, &nValue);
// Error handling routine is omitted

// Set template size to nValue
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_TEMPLATE_SIZE, &nValue);
// Error handling routine is omitted

// Set use SIF to nValue
ufs_res = UFS_SetParameter(hScanner, UFS_PARAM_USE_SIF, &nValue);
// Error handling routine is omitted
```

Return Values([refer to return values](#))

UFS_IsSensorOn

Checks whether a scanner is connected or not.

```
UFS_STATUS UFS_API UFS_IsSensorOn(  
    HUFScanner hScanner,  
    int* pbSensorOn );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pbSensorOn [out]* : Receive the status of specified scanner object; 1: the scanner is connected, 0: the scanner is disconnected

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int bSensorOn;  
// Get hScanner handle  
  
ufs_res = UFS_IsSensorOn(hScanner, &bSensorOn);  
if (ufs_res == UFS_OK)  
{  
    // UFS_IsSensorOn is succeeded  
}  
else  
{  
    // UFS_IsSensorOn is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_IsFingerOn

Checks whether a finger is placed on a scanner or not.

```
UFS_STATUS UFS_API UFS_IsFingerOn(  
    HUFScanner hScanner,  
    int* pbFingerOn );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pbFingerOn [out]* : Checks if a finger is placed on the specified scanner; 1: a finger is on the scanner, 0: a finger is not on the scanner

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
int bFingerOn;  
  
// Get hScanner handle  
  
ufs_res = UFS_IsFingerOn(hScanner, &bFingerOn);  
  
if (ufs_res == UFS_OK)  
{  
    // UFS_IsFingerOn is succeeded  
}  
else  
{  
    // UFS_IsFingerOn is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_CaptureSingleImage

Captures single image. Captured image is stored to the internal buffer.

```
UFS_STATUS UFS_API UFS_CaptureSingleImage(  
    HUFScanner hScanner, );
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
// Get hScanner handle  
  
ufs_res = UFS_CaptureSingleImage(hScanner);  
  
if (ufs_res == UFS_OK)  
{  
    // UFS_CaptureSingleImage is succeeded  
}  
else  
{  
    // UFS_CaptureSingleImage is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_StartCapturing

Starts capturing. The capture is going on until the specified time exceeds.

```
UFS_STATUS UFS_API UFS_StartCapturing(  
    HUFScanner hScanner,  
    UFS_CAPTURE_PROC* pCaptureProc,  
    void* pParam );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pCaptureProc [in]* : Handle to the UFS_CAPTURE_PROC function which receives capture events
- *pParam [in]* : Pointer to the capture callback data which will be transmitted with a capture callback event

Examples

```
// Define capture procedure  
int UFS_CALLBACK CaptureProc(HUFScanner hScanner, int bFingerOn, unsigned  
char* pImage, int nWidth, int nHeight, int nResolution, void* pParam)  
{  
    // ...  
}  
  
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
void* pParam;  
  
// Get hScanner handle  
  
// Assign pParam, for example, application data  
  
ufs_res = UFS_StartCapturing(hScanner, CaptureProc, pParam);  
  
if (ufs_res == UFS_OK)  
{  
    // UFS_StartCapturing is succeeded  
}  
else  
{  
    // UFS_StartCapturing is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_StartAutoCapture

Starts the automatic capture. Currently this function is working for Suprema SFR600(BioMini Slim) only.

```
UFS_STATUS UFS_API UFS_StartAutoCapture(  
    HUFSScanner hScanner,  
    UFS_CAPTURE_PROC* pCaptureProc,  
    void* pParam );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pCaptureProc [in]* : Handle to the UFS_CAPTURE_PROC function which receives capture events
- *pParam [in]* : Pointer to the capture callback data which will be transmitted with a capture callback event

Examples

```
// Define capture procedure  
int UFS_CALLBACK CaptureProc(HUFSScanner hScanner, int bFingerOn, unsigned  
char* pImage, int nWidth, int nHeight, int nResolution, void* pParam)  
{  
    // ...  
}  
  
UFS_STATUS ufs_res;  
HUFSScanner hScanner;  
void* pParam;  
  
// Get hScanner handle  
  
// Assign pParam, for example, application data  
  
ufs_res = UFS_StartAutoCapture(hScanner, CaptureProc, pParam);  
if (ufs_res == UFS_OK)  
{  
    // UFS_StartAutoCapture is succeeded  
}  
else  
{  
    // UFS_StartAutoCapture is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_IsCapturing

Checks if the specified scanner is running to capture which is started by [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#)

```
UFS_STATUS UFS_API UFS_IsCapturing(  
    HUFScanner hScanner,  
    int* pbCapturing );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pbCapturing [out]* : Checks if the specified scanner is running capturing; 1: the capture is running, 0: the capture is not running

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int bCapturing;  
  
// Get hScanner handle  
  
ufs_res = UFS_IsCapturing(hScanner, &bCapturing);  
if (ufs_res == UFS_OK)  
{  
    //UFS_IsCapturing is succeeded  
}  
else  
{  
    // UFS_IsCapturing is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_AbortCapturing

Aborts capturing which is started by [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#).

```
UFS_STATUS UFS_API UFS_AbortCapturing(  
    HUFScanner hScanner, );
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
// Get hScanner handle  
  
// Start capturing  
  
ufs_res = UFS_AbortCapturing(hScanner);  
if (ufs_res == UFS_OK)  
{  
    // UFS_AbortCapturing is succeeded  
}  
else  
{  
    // UFS_AbortCapturing is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_Extract

Extracts a template from the stored image buffer which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#).

```
UFS_STATUS UFS_API UFS_Extract(  
    HUFScanner hScanner,  
    unsigned char* pTemplate,  
    int* pnTemplateSize,  
    int* pnEnrollQuality);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplate [out]* : Pointer to the template array; The array must be allocated in advance
- *pnTemplateSize [out]* : Receives the size (in bytes) of pTemplate
- *pnEnrollQuality [out]* : Receives the quality of enrollment; Quality value ranges from 1 to 100. Typically this value should be above 30 for further processing such as enroll and matching. Especially in case of enrollment, the use of good quality image (above 50) is highly recommended.

Examples

```
// Template size can be controlled by using UFS_SetParameter function  
// Default value is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
unsigned char Template[MAX_TEMPLATE_SIZE];  
int TemplateSize;  
int nEnrollQuality;  
  
// Get hScanner handle  
  
ufs_res = UFS_Extract(hScanner, Template, &TemplateSize, &nEnrollQuality);  
if (ufs_res == UFS_OK)  
{  
    // UFS_Extract is succeeded  
}  
else  
{  
    // UFS_Extract is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_ExtractEx

Extracts a template from the stored image buffer which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#). This is extended version of [UFS_Extract](#) function to accommodate large size template.

```
UFS_STATUS UFS_API UFS_ExtractEx(  
    HUFScanner hScanner,  
    int* nBufferSize,  
    unsigned char* pTemplate,  
    int* pnTemplateSize,  
    int* pnEnrollQuality);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nBufferSize [in]* : Template buffer size
- *pTemplate [out]* : Pointer to the template array; The array must be allocated in advance
- *pnTemplateSize [out]* : Receives the size (in bytes) of pTemplate
- *pnEnrollQuality [out]* : Receives the quality of enrollment; Quality value ranges from 1 to 100. Typically this value should be above 30 for further processing such as enroll and matching. Especially in case of enrollment, the use of good quality image (above 50) is highly recommended.

Examples

```
// Template size can be controlled by using UFS_SetParameter function  
// Default value is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
unsigned char Template[MAX_TEMPLATE_SIZE];  
int TemplateSize;  
int nEnrollQuality;  
  
// Get hScanner handle  
  
ufs_res = UFS_ExtractEx(hScanner, MAX_TEMPLATE_SIZE, Template,  
&TemplateSize, &nEnrollQuality);  
if (ufs_res == UFS_OK)  
{  
    // UFS_ExtractEx is succeeded  
}  
else  
{  
    // UFS_ExtractEx is failed
```

```
// Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_SetEncryptionKey

Sets encryption key.

```
UFS_STATUS UFS_API UFS_SetEncryptionKey(  
    HUFScanner hScanner,  
    unsigned char* pKey );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pKey[out]* : Pointer to the 32 bytes key array; default key is first byte is 1 and second to 32th byte are all 0

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
unsigned char UserKey[32];  
  
// Get hScanner handle  
// Generate 32 byte encryption key to UserKey  
  
ufs_res = UFS_SetEncryptionKey(hScanner, UserKey);  
if (ufs_res == UFS_OK)  
{  
    // UFS_SetEncryptionKey is succeeded  
}  
else  
{  
    // UFS_SetEncryptionKey is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_EncryptTemplate

Encrypts template.

```
UFS_STATUS UFS_API UFS_EncryptTemplate(  
    HUFScanner hScanner,  
    unsigned char* pTemplateInput,  
    int nTemplateInputSize,  
    unsigned char* pTemplateOutput,  
    int* pnTemplateOutputSize);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplate [in]* : Pointer to input template data
- *nTemplateInputSize [in]* : Input template size
- *pTemplateOutput [out]* : Pointer to encrypted template data
- *pnTemplateOutputSize [in / out]* : Inputs allocated size of encrypted template data; Receives output template size

Examples

```
// Assume template size is 384 bytes  
#define MAX_TEMPLATE_SIZE 384  
  
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
unsigned char TemplateInput[MAX_TEMPLATE_SIZE];  
unsigned char TemplateOutput[MAX_TEMPLATE_SIZE];  
int TemplateInputSize;  
int TemplateOutputSize;  
  
// Get hScanner handle  
// Get an input template to encrypt, TemplateInput and TemplateInputSize  
// Set output template buffer size TemplateOutputSize = MAX_TEMPLATE_SIZE;  
  
ufs_res = UFS_EncryptTemplate(hScanner, TemplateInput, TemplateInputSize,  
    TemplateOutput, &TemplateOutputSize);  
if (ufs_res == UFS_OK)  
{  
    // UFS_EncryptTemplate is succeeded  
}  
else  
{  
    // UFS_EncryptTemplate is failed  
    // Use UFS_GetErrorString function to show error string  
}
```


Return Values([refer to return values](#))

UFS_DecryptTemplate

Decrypts template.

```
UFS_STATUS UFS_API UFS_DecryptTemplate(  
    HUFScanner hScanner,  
    unsigned char* pTemplateInput,  
    int nTemplateInputSize,  
    unsigned char* pTemplateOutput,  
    int* pnTemplateOutputSize);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplateInput [in]* : Pointer to input template data(encrypted)
- *nTemplateInputSize [in]* : Input template size
- *pTemplateOutput [out]* : Pointer to output template data
- *pnTemplateOutputSize [in / out]* : Inputs allocated size of output template data; Receives output template size

Examples

```
// Assume template size is 384 bytes  
#define MAX_TEMPLATE_SIZE 384  
  
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
unsigned char TemplateInput[MAX_TEMPLATE_SIZE];  
unsigned char TemplateOutput[MAX_TEMPLATE_SIZE];  
int TemplateInputSize;  
int TemplateOutputSize;  
  
// Get hScanner handle  
// Get an encrypted template, TemplateInput and TemplateInputSize  
// Set output template buffer size  
  
TemplateOutputSize = MAX_TEMPLATE_SIZE;  
  
ufs_res = UFS_DecryptTemplate(hScanner, TemplateInput, TemplateInputSize,  
    TemplateOutput, &TemplateOutputSize);  
if (ufs_res == UFS_OK)  
{  
    // UFS_DecryptTemplate is succeeded  
}  
else  
{  
    // UFS_DecryptTemplate is failed
```

```
// Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferInfo

Gets the information of the capture image buffer.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBufferInfo (
    HUFSScanner hScanner,
    int* pnWidth,
    int* pnHeight,
    int* pnResolution );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnWidth [out]* : Receives the width of the capture image buffer
- *pnHeight [out]* : Receives the height of the capture image buffer
- *pnResolution [out]* : Receives the resolution of the capture image buffer

Examples

```
UFS_STATUS ufs_res;
HUFSScanner hScanner;
int nWidth;
int nHeight;
int nResolution;

// Get hScanner handle

ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,
&nResolution);
if (ufs_res == UFS_OK)
{
    // UFS_GetCaptureImageBufferInfo is succeeded
}
else
{
    // UFS_GetCaptureImageBufferInfo is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values[\(refer to return values\)](#)

UFS_GetCaptureImageBuffer

Copies the capture image buffer to the specified image data array.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBuffer(  
    HUFScanner hScanner,  
    unsigned char* pImageData );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to image data array; The array must be allocated bigger than the size of capture image buffer in advance

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
int nResolution;  
unsigned char* pImageData  
  
// Get hScanner handle  
// Get capture image buffer information  
  
ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,  
&nResolution);  
  
// Error handling routine is omitted  
// Allocate image buffer  
  
pImageData = (unsigned char*)malloc(nWidth * nHeight * sizeof(unsigned  
char));  
  
ufs_res = UFS_GetCaptureImageBuffer(hScanner, pImageData);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetCaptureImageBuffer is succeeded  
}  
else  
{  
    // UFS_GetCaptureImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}  
  
// Free image buffer after usage  
free(pImageBuffer)
```

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferToBMPImageBuffer

Copies the capture image buffer to the specified image data of bmp format.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBufferToBMPImageBuffer(  
    HUFScanner hScanner,  
    unsigned char* pImageData,  
    int* pImageDataLength );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to bmp image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *pImageDataLength [out]* : pointer to bmp image data size

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
int nResolution;  
int nBmpHeaderSize;  
unsigned char* pBmpImageBuf;  
int nBmpImageBufSize;  
  
// Get hScanner handle  
  
ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,  
&nResolution);  
// Error handling routine is omitted  
  
nBmpHeaderSize = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +  
sizeof(RGBQUAD);  
  
// Allocate bmp image buffer  
pBmpImageBuf = (unsigned char*)malloc(nWidth * nHeight + nBmpHeaderSize);  
  
ufs_res = UFS_GetCaptureImageBufferToBMPImageBuffer(hScanner, pBmpImageBuf,  
&nBmpImageBufSize);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetCaptureImageBufferToBMP is succeeded  
}  
else  
{
```

```
// UFS_GetCaptureImageBufferToBMP is failed  
// Use UFS_GetErrorString function to show error string  
}  
  
// Free image buffer after usage  
free(pBmpImageBuf);
```

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferTo19794_4ImageBuffer

Copies the capture image buffer to the specified image data of 19794_4 format.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBufferTo19794_4ImageBuffer(  
    HUFScanner hScanner,  
    unsigned char* pImageData,  
    int* pImageDataLength );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to 19794_4 format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *pImageDataLength [out]* : pointer to 19794_4 format image data size

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
int nResolution;  
unsigned char* pConvertedImageBuf;  
int nConvertedImageBufSize;  
  
// Get hScanner handle  
  
ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,  
&nResolution);  
// Error handling routine is omitted  
  
pConvertedImageBuf = (unsigned char*)malloc(nWidth * nHeight);  
  
ufs_res = UFS_GetCaptureImageBufferTo19794_4ImageBuffer(hScanner,  
pConvertedImageBuf, &nConvertedImageBufSize);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetCaptureImageBufferTo19794_4ImageBuffer is succeeded  
}  
else  
{  
    // UFS_GetCaptureImageBufferTo19794_4ImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}  
  
// Free image buffer after usage  
free(pConvertedImageBuf)
```

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferToWSQImageBuffer

Copies the capture image buffer to the specified image data of the WSQ format.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBufferToWSQImageBuffer(  
    HUFScanner hScanner,  
    const float ratio,  
    unsigned char* wsqData,  
    int* wsqDataLen );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *wsqData [out]* : Pointer to WSQ format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *wsqDataLen [out]* : pointer to WSQ format image data size

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
int nResolution;  
unsigned char* pConvertedImageBuf;  
int nConvertedImageBufSize;  
float nRatio = 0.75;  
  
// Get hScanner handle  
  
ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,  
&nResolution);  
// Error handling routine is omitted  
  
pConvertedImageBuf = (unsigned char*)malloc(nWidth * nHeight);  
  
ufs_res = UFS_GetCaptureImageBufferToWSQImageBuffer(hScanner, nRatio,  
pConvertedImageBuf, &nConvertedImageBufSize);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetCaptureImageBufferToWSQImageBuffer is succeeded  
}  
else  
{  
    // UFS_GetCaptureImageBufferToWSQImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

```
// Free image buffer after usage  
free(pConvertedImageBuf)
```

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferToWSQImageBufferVar

Copies the capture image buffer (cropped or expanded to the specified size) to the target image data buffer of the WSQ format.

```
UFS_STATUS UFS_API UFS_GetCaptureImageBufferToWSQImageBufferVar(  
    HUFScanner hScanner,  
    Const float ratio,  
    unsigned char* wsqData,  
    int* wsqDataLen,  
    int nWidth,  
    int nHeight );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *wsqData [out]* : Pointer to WSQ format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *wsqDataLen [out]* : pointer to WSQ format image data size
- *nWidth [in]* : Width to resize the capture image
- *nHeight [in]* : Height to resize the capture image

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
unsigned char* pConvertedImageBuf;  
int nConvertedImageBufSize;  
float nRatio = 0.75;  
  
// Get hScanner handle  
  
// Get image size to resize(nWidth, nHeight)  
  
pConvertedImageBuf = (unsigned char*)malloc(nWidth * nHeight);  
  
ufs_res = UFS_GetCaptureImageBufferToWSQImageBufferVar(hScanner, nRatio,  
pConvertedImageBuf, &nConvertedImageBufSize, nWidth, nHeight);  
  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetCaptureImageBufferToWSQImageBufferVar is succeeded  
}  
else  
{
```

```
// UFS_GetCaptureImageBufferToWSQImageBufferVar is failed  
// Use UFS_GetErrorString function to show error string  
}  
  
// Free image buffer after usage  
free(pConvertedImageBuf)
```

Return Values([refer to return values](#))

UFS-DecompressWSQBMP

Decompress WSQ file and save to BMP file.

```
UFS_STATUS UFS_API UFS-DecompressWSQBMP(  
    HUFScanner hScanner,  
    char* wsqFile,  
    char* bmpFile );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *wsqFile [in]* : Specifies file name to get wsq data buffer
- *bmpFile [in]* : Specifies file name to save image buffer

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
char szWsqFileName[128];  
char szBmpFileName[128];  
  
// Get hScanner handle  
// Get WSQ file name to save bmp file  
// Get Bmp file from the WSQ file  
  
ufs_res = UFS-DecompressWSQBMP (hScanner, szWsqFileName, szBmpFileName);  
if (ufs_res == UFS_OK)  
{  
    // UFS-DecompressWSQBMP is succeeded  
}  
else  
{  
    // UFS-DecompressWSQBMP is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS-DecompressWSQBMPMem

Decompress WSQ buffer and save to image data of bmp format.

```
UFS_STATUS UFS_API UFS-DecompressWSQBMPMem(  
    HUFScanner hScanner,  
    unsigned char* wsqBuffer,  
    int wsqBufferLen,  
    unsigned char* bmpBuffer,  
    int* bmpBufferLen );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *wsqBuffer [in]* : Pointer to WSQ format image data
- *wsqBufferLen [in]* : Size of WSQ format image data
- *bmpBuffer [out]* : Pointer to bmp image data; The array must be allocated bigger than the size of capture image buffer in advance.
- *bmpBufferLen [out]* : pointer to bmp image data size

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nWidth;  
int nHeight;  
int nResolution;  
int nBmpHeaderSize;  
unsigned char* pWsqBuffer;  
int nWsqBufferLen;  
unsigned char* pBmpBuffer;  
int nBmpBufferLen;  
  
// Get hScanner handle  
// Get WSQ data (ex.UFS_GetCaptureImageBufferToWSQImageBuffer)  
  
ufs_res = UFS_GetCaptureImageBufferInfo(hScanner, &nWidth, &nHeight,  
&nResolution);  
// Error handling routine is omitted  
  
nBmpHeaderSize = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +  
sizeof(RGBQUAD);  
  
pBmpBuffer = (unsigned char*)malloc(nWidth * nHeight * nBmpHeaderSize);  
  
ufs_res = UFS-DecompressWSQBMPMem(hScanner, pWsqBuffer, nWsqBufferLen,  
pBmpBuffer, nBmpBufferLen);  
if (ufs_res == UFS_OK)
```



```
{
    // UFS_DecompressWSQBMPMem is succeeded
}
else
{
    // UFS_DecompressWSQBMPMem is failed
    // Use UFS_GetErrorString function to show error string
}

// Free image buffer after usage
free(pImageBuffer);
free(pWsqBuffer);
```

Return Values([refer to return values](#))

UFS_DrawCaptureImageBuffer

Draws the fingerprint image which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#). This function is not supported on java.

```
UFS_STATUS UFS_API UFS_DrawCaptureImageBuffer(  
    HUFScanner hScanner,  
    HDC hDC,  
    int nLeft,  
    int nTop,  
    int nRight,  
    int nBottom,  
    int bCore );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *hDC [in]* : Handle to the DC where the fingerprint image is drawn
- *nLeft [in]* : Specifies the logical x-coordinate of the upper-left corner of the rectangle
- *nTop [in]* : Specifies the logical y-coordinate of the upper-left corner of the rectangle
- *nRight [in]* : Specifies the logical x-coordinate of the lower-right corner of the rectangle
- *nBottom [in]* : Specifies the logical y-coordinate of the lower-right corner of the rectangle
- *bCore [in]* : Specifies whether the core of fingerprint is drawn or not

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
HDC hDC;  
int nLeft;  
int nTop;  
int nRight;  
int nBottom;  
int bCore;  
  
// Get hScanner handle  
// Get HDC and determine rectangle to draw image, hDC, nLeft, nTop, nRight, nBottom  
// Determine core to be drawn, bCore  
  
ufs_res = UFS_DrawCaptureImageBuffer(hScanner, hDC, nLeft, nTop, nRight, nBottom, bCore);  
if (ufs_res == UFS_OK)  
{  
    // UFS_DrawCaptureImageBuffer is succeeded  
}  
else  
{
```

```
// UFS_DrawCaptureImageBuffer is failed  
// Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_DrawFeatureBuffer

Draws the fingerprint image which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#). This function is not supported on java. And should be called after extraction from last captured fingerprint image. If extraction is not performed from the last captured image, this function will not draw the feature in image frame.

```
UFS_STATUS UFS_API UFS_DrawFeatureBuffer(  
    HUFScanner hScanner,  
    HDC hDC,  
    int nLeft,  
    int nTop,  
    int nRight,  
    int nBottom,  
    int bCore );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *hDC [in]* : Handle to the DC where the fingerprint image is drawn
- *nLeft [in]* : Specifies the logical x-coordinate of the upper-left corner of the rectangle
- *nTop [in]* : Specifies the logical y-coordinate of the upper-left corner of the rectangle
- *nRight [in]* : Specifies the logical x-coordinate of the lower-right corner of the rectangle
- *nBottom [in]* : Specifies the logical y-coordinate of the lower-right corner of the rectangle
- *bCore [in]* : Specifies whether the core of fingerprint is drawn or not

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
HDC hDC;  
int nLeft;  
int nTop;  
int nRight;  
int nBottom;  
int bCore;  
  
// Get hScanner handle  
// Get HDC and determine rectangle to draw image, hDC, nLeft, nTop, nRight, nBottom  
// Determine core to be drawn, bCore  
  
ufs_res = UFS_DrawFeatureBuffer(hScanner, hDC, nLeft, nTop, nRight, nBottom, bCore);  
if (ufs_res == UFS_OK)  
{  
    // UFS_DrawFeatureBuffer is succeeded  
}
```

```
else
{
    // UFS_DrawFeatureBuffer is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values([refer to return values](#))

UFS_SaveCaptureImageBufferToBMP

Saves the capture image buffer to the specified file of the bitmap format.

```
UFS_STATUS UFS_API UFS_SaveCaptureImageBufferToBMP(  
    HUFScanner hScanner,  
    char* szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szFileName [in]* : Specifies file name to save image buffer

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
char szFileName[128];  
  
// Get hScanner handle  
// Get file name, szFileName  
  
ufs_res = UFS_SaveCaptureImageBufferToBMP(hScanner, szFileName);  
if (ufs_res == UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferToBMP is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferToBMP is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to [return values](#))

UFS_SaveCaptureImageBufferTo19794_4

Saves the capture image buffer to the specified file of the 19794_4 format.

```
UFS_STATUS UFS_API UFS_SaveCaptureImageBufferTo19794_4(  
    HUFScanner hScanner,  
    char* szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szFileName [in]* : Specifies file name to save image buffer

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
char szFileName[128];  
  
// Get hScanner handle  
// Get file name, szFileName  
  
ufs_res = UFS_SaveCaptureImageBufferTo19794_4(hScanner, szFileName);  
if (ufs_res == UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferTo19794_4 is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferTo19794_4 is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_SaveCaptureImageBufferToWSQ

Saves the capture image buffer to the specified file of the WSQ format.

```
UFS_STATUS UFS_API UFS_SaveCaptureImageBufferToWSQ(  
    HUFScanner hScanner,  
    const float ratio,  
    char* szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *szFileName [in]* : Specifies file name to save image buffer

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
char szFileName[128];  
  
// Get hScanner handle  
// Get file name; szFileName  
  
ufs_res = UFS_SaveCaptureImageBufferToWSQ(hScanner, ratio, szFileName);  
if (ufs_res == UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferToWSQ is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferToWSQ is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_SaveCaptureImageBufferToWSQVar

Saves the capture image buffer (cropped or expanded to the specified size) to the target file of the WSQ format.

```
UFS_STATUS UFS_API UFS_SaveCaptureImageBufferToWSQVar(  
    HUFScanner hScanner,  
    const float ratio,  
    char* szFileName,  
    int nWidth,  
    int nHeight );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *szFileName [in]* : Specifies file name to save image buffer
- *nWidth [in]* : Width to resize the capture image
- *nHeight [in]* : Height to resize the capture image

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
char szFileName[128];  
int nWidth;  
int nHeight;  
  
// Get hScanner handle  
// Get file name, szFileName  
// Get size of capture image to resize; nWidth,nHeight  
  
ufs_res = UFS_SaveCaptureImageBufferToWSQVar(hScanner, ratio, szFileName,  
nWidth, nHeight);  
if (ufs_res == UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferToWSQVar is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferToWSQVar is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_ClearCaptureImageBuffer

Clears the capture image buffer stored to the internal buffer.

```
UFS_STATUS UFS_API UFS_ClearCaptureImageBuffer(  
    HUFScanner hScanner );
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
  
// Get hScanner handle  
  
ufs_res = UFS_ClearCaptureImageBuffer(hScanner);  
if (ufs_res == UFS_OK)  
{  
    // UFS_ClearCaptureImageBuffer is succeeded  
}  
else  
{  
    // UFS_ClearCaptureImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetErrorString

Gets the error string for specified [UFS_STATUS](#) value.

```
UFS_STATUS UFS_API UFS_GetErrorString(  
    UFS_STATUS res,  
    char* szErrorString );
```

Parameters

- *res [in]* : Status return value
- *szErrorString [out]* : Receives error string

Examples

```
UFS_STATUS ufs_res;  
char strError[128];  
  
// Get status return code, ufs_res  
  
ufs_res = UFS_GetErrorString(ufs_res, strError);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetErrorString is succeeded  
}  
else  
{  
    // UFS_GetErrorString is failed  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetTemplateType

Gets the template type value.

```
UFS_STATUS UFS_API UFS_GetTemplateType(  
    HUFScanner hScanner,  
    int* nTemplateType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTemplateType [out]* : Receives the parameter value of specified parameter type; 'pValue' must point to adequate type that is matched with the parameter type

Template type	Code	Description
UFS_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFS_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFS_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
int nTemplateType;  
// Get hScanner handle  
  
ufs_res = UFS_GetTemplateType(hScanner, &nTemplateType);  
// Error handling routine is omitted
```

Return Values(refer to return values)

UFS_SetTemplateType

Sets the template type value.

```
UFS_STATUS UFS_API UFS_SetTemplateType(  
    HUFSscanner hScanner,  
    int nTemplateType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTemplateType [in]* : Parameter type; one of template type

Template type	Code	Description
UFS_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFS_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFS_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
UFS_STATUS ufs_res;  
HUFSscanner hScanner;  
int nTemplateType;  
// Get hScanner handle  
  
nTemplateType = UFS_TEMPLATE_TYPE_SUPREMA;  
  
ufs_res = UFS_SetTemplateType(hScanner, nTemplateType);  
// Error handling routine is omitted
```

Return Values(refer to return values)

UFS_SelectTemplate

Selects n number of good templates from m number of input templates.

```
UFS_STATUS UFS_API UFS_SelectTemplate(  
    HUFSScanner hScanner,  
    unsigned char** ppTemplateInput,  
    int* pnTemplateInputSize,  
    int nTemplateInputNum,  
    unsigned char** ppTemplateOutput,  
    int* pnTemplateOutputSize,  
    int nTemplateOutputNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ppTemplateInput [in]* : Array pointer to the input template arrays
- *pnTemplateInputSize [in]* : Array pointer to input templates'size
- *nTemplateInputNum [in]* : Number of input templates
- *ppTemplateOutput [out]* : Array pointer to the output template arrays
- *pnTemplateOutputSize [out]* : Array pointer to the output templates'size
- *nTemplateOutputNum [in]* : Number of output templates; should be less than input template number by more than one

Examples

```
unsigned char Template[MAX_TEMPLATE_SIZE];  
int TemplateSize;  
int nEnrollQuality;  
UFS_STATUS ufs_res;  
int i = ;  
// sample number  
int inputNum = 4;  
int outputNum = 2;  
  
While(1)  
{  
    // capture a fingerprint image  
    .  
    .  
    ufs_res = UFS_Extract(hScanner, Template, &TemplateSize,  
&nEnrollQuality);  
  
    .  
    .  
    // if UFS_Extract is succeed  
    memcpy(InputTemplateArray[i], Template, TemplateSize);  
    InputTemplateSizeArray[i] = TemplateSize;
```

```

    i++;
    if(i == inputNum)
        break;
}

ufs_res = UFS_SelectTemplate(hScanner, InputTemplateArray,
InputTemplateSizeArray, inputNum, OutputTemplateArray,
OutputTemplateSizeArray, outputNum);

// UFS_SelectTemplate is succeed
if (ufs_res == UFS_OK)
{
    // If you want to enroll the output templates, move OutputTemplateArray
    and OutputTemplateSizeArray data to your template array for enrollment or
    database.
}
// select template function has error.
else
{
    // ...
}

```

Return Values([refer to return values](#))

UFS_SelectTemplateEx

Selects n number of good templates from m number of input templates. This is extended version of [UFS_SelectTemplate](#) function to accommodate large size template.

```
UFS_STATUS UFS_API UFS_SelectTemplateEx(  
    HUFScanner hScanner,  
    int nBufferSize,  
    unsigned char** ppTemplateInput,  
    int* pnTemplateInputSize,  
    int nTemplateInputNum,  
    unsigned char** ppTemplateOutput,  
    int* pnTemplateOutputSize,  
    int nTemplateOutputNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nBufferSize [in]* : Template buffer size
- *ppTemplateInput [in]* : Array pointer to the input template arrays
- *pnTemplateInputSize [in]* : Array pointer to the input templates'size
- *nTemplateInputNum [in]* : Number of input templates
- *ppTemplateOutput [out]* : Array pointer to the output template arrays
- *pnTemplateOutputSize [out]* : Array pointer to the output templates'size
- *nTemplateOutputNum [in]* : Number of output templates; should be less than input template number by more than one

Examples

```
unsigned char Template[MAX_TEMPLATE_SIZE];  
int TemplateSize;  
int nEnrollQuality;  
UFS_STATUS ufs_res;  
int i = ;  
// sample number  
int inputNum = 4;  
int outputNum = 2;  
  
While(1)  
{  
    // capture a fingerprint image  
    .  
    .  
    ufs_res = UFS_Extract(hScanner, Template, &TemplateSize,  
&nEnrollQuality);  
    .  
    .
```



```

    // if UFS_Extract is succeed
    memcpy(InputTemplateArray[i], Template, TemplateSize);
    InputTemplateSizeArray[i] = TemplateSize;
    i++; if(i == inputNum)
        break;
}

ufs_res = UFS_SelectTemplateEx(hScanner, MAX_TEMPLATE_SIZE,
InputTemplateArray, InputTemplateSizeArray, inputNum, OutputTemplateArray,
OutputTemplateSizeArray, outputNum);

// UFS_SelectTemplate is succeed
if (ufs_res == UFS_OK)
{
    // If you want to enroll the output templates, move OutputTemplateArray
    and OutputTemplateSizeArray data to your template array for enrollment or
    database.
}
// select template function has error.
else
{
    // ...
}

```

Return Values(refer to return values)

UFS_GetFPQuality

Calculates the quality score of an image as defined in NISTIR 7151: FingerPrint Image Quality. The score would be between 1(excellent) and 5(poor).

```
UFS_STATUS UFS_API UFS_GetFPQuality(  
    HUFScanner hScanner,  
    unsigned char* pFPImage,  
    int nWidth,  
    int nHeight,  
    int* pnFPQuality );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pFPImage [in]* : Raw capture image data
- *nWidth [in]* : Width of capture image data
- *nHeight [in]* : Height of capture image data
- *pnFPQuality [in]* : NIST quality score of image data

Examples

```
UFS_STATUS ufs_res;  
HUFScanner hScanner;  
unsigned char *pCaptImageBuf;  
int nCaptImageWidth;  
int nCaptImageHeight;  
int nFPQuality;  
  
// Capture a fingerprint image  
// Get Capture Image Data; pCaptImageBuf, nCaptImageWidth, nCaptImageHeight  
  
ufs_res = UFS_GetFPQuality(hScanner, pCaptImageBuf, nCaptImageWidth,  
nCaptImageWidth, &nFPQuality);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetFPQuality is succeeded  
}  
else  
{  
    // UFS_GetFPQuality is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_GetFeatureNumber

Get the number of Minutiae from the template data.

```
UFS_STATUS UFS_API UFS_GetFeatureNumber (
    HUFSScanner hScanner,
    unsigned char* pTemplate,
    int nTemplateSize,
    int* pnFeatureNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplate [in]* : Template data
- *nTemplateSize [in]* : Size of template data
- *pnFeatureNum [out]* : The number of minutiae from pTemplate

Examples

```
unsigned char Template[MAX_TEMPLATE_SIZE];
int TemplateSize;
int nFeatureNum;
int nEnrollQuality;
UFS_STATUS ufs_res;
HUFSScanner hScanner;
// Capture a fingerprint image

ufs_res = UFS_Extract(hScanner, Template, &TemplateSize, &nEnrollQuality);
// Error handling routine is omitted

ufs_res = UFS_GetFeatureNumber(hScanner, Template, TemplateSize,
    &nFeatureNum);

if (ufs_res == UFS_OK)
{
    // UFS_GetFeatureNumber is succeeded
}
else
{
    // UFS_GetFeatureNumber is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values[\(refer to return values\)](#)

UFS_EnrollUI

Generate the fingerprint enrollment dialog. This function can be called after executing UFS_Init. Enrolling a fingerprint is extracting a template from finger and saving the template. Below sample's UFS_EnrollUI function captures a fingerprint image after setting the template type. And extracts a template from captured fingerprint image. The extracted template will be saved in a specific template array, which is a parameter of the UFS_EnrollUI function. It supported only for Windows environment.

*Constraints

- You should have 'img' folder to use graphical backgrounds and buttons. The application uses the img folder should be at the upper level folder. For example, if the application is at the /bin/sample, 'img' folder should be at the location of /bin/sample/img.
- Enrollment UI is based on COM interface. Thus you should register dll file before use. You can use the pre-coded script (register_enrollui.bat) to register the dll file, or simple type the command 'regsvr32.exe IEnrollUI.dll' at the command prompt.

```
UFS_STATUS UFS_API UFS_EnrollUI(  
    HUFSscanner hScanner,  
    int nTimeout,  
    int nOptions,  
    BYTE* pUF_FIR_Buf,  
    int* pUF_FIR_Buf_Len,  
    BYTE* pISO_FIR_Buf,  
    int* pISO_FIR_Buf_Len,  
    char* pImages_Path);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTimeout [in]* : Timeout of the capture
- *nOptions [in]* : Options for enrollment. Matching level, image Quality, number of fingerprints for enrollment, number of templates per finger
- *pUF_FIR_Buf [out]* : Pointer to the byte array for suprema template. This data pointer is assigned by maximum 1024*20
- *pUF_FIR_Buf_Len [out]* : Pointer to the int array for length of suprema template buffer
- *pISO_FIR_Buf [out]* : Pointer to the byte array for ISO template. This data pointer is assigned by maximum 1024 * 20
- *pISO_FIR_Buf_Len [out]* : Pointer to the int array for length of ISO template buffer
- *pImages_Path [in]* : Path to captured images to be saved. If NULL value is passed, nothing will be saved

Examples

```
// (1) initialize the buffer  
BYTE* pUFBuf = new BYTE[1024*20];  
memset(pUFBuf, , 1024*20);  
int* pUFBufSize = new int[20];  
memset(pUFBufSize, , 20);
```

```

BYTE* pISOBuf = new BYTE[1024*20];
memset(pISOBuf, , 1024*20);
int* pISOBufSize = new int[20];
memset(pISOBufSize, , 20);

int nFingersToEnroll = 10;
int nTemplatesPerFinger = 2;

// (2) set the options
int nOptions = UF_PACK_SECURITY(m_nSecurityLevel+1) |
               UF_PACK_QUALITY(m_quality) |
               UF_PACK_NFINGERS(nFingersToEnroll) |
               UF_PACK_NTEMPLATES(nTemplatesPerFinger); //

// (3) execute the enrollment ui api
UFS_EnrollUI(hScanner, m_nTimeout, nOptions, pUFBuf, pUFBufSize, pISOBuf,
pISOBufSize, "c:\\");

// (4) get the templates buffer
memcpy(m_pTemplateBuf, pUFBuf, 1024*20*sizeof(BYTE));
memcpy(m_pTemplateBufSize, pUFBufSize, 20*sizeof(int));

// (5) release buffer
delete[] pUFBuf;
delete[] pUFBufSize;
delete[] pISOBuf;
delete[] pISOBufSize;

```

Return Values([refer to return values](#))

UFS_VerifyUI

Generate the fingerprint verification dialog. This function can be called after executing UFS_Init and UFS_EnrollUI. Two fingerprints can be verified whether they are matched or not. Below sample's UFS_VerifyUI function captures a fingerprint image and extracts a template from the image. And execute 1:1 matching using extracted template and templates enrolled from UFS_EnrollUI.

*Constraints

- Refer to the UFS_EnrollUI constraints.

```
UFS_STATUS UFS_API UFS_VerifyUI(  
    HUFSScanner hScanner,  
    int nTimeout,  
    int nOptions,  
    int nFPTemplateType,  
    BYTE* pFIR_BUF,  
    int* pFIR_Buf_Len,  
    char* pImages_Path,  
    int* nFingerIndex);
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTimeout [in]* : Timeout of the capture
- *nOptions [in]* : Options for enrollment. Matching level, image Quality, number of fingerprints for enrollment, number of templates per finger
- *nFPTemplateType [in]* : Template type for matching enrolled templates with captured fingerprint
- *pFIR_Buf [out]* : Pointer to the byte array for template
- *pFIR_Buf_Len [out]* : Pointer to the int array for length of template buffer
- *pImages_Path [in]* : Path to captured images
- *nFingerIndex [in]* : Matched finger index from enrolled templates. If this value is -1, the matching result is failed

Examples

```
int nFingersToEnroll = 10;  
int nTemplatesPerFinger = 2;  
  
// (1) set the options  
int nOptions = UF_PACK_SECURITY(m_nSecurityLevel+1) |  
               UF_PACK_QUALITY(m_quality) |  
               UF_PACK_NFINGERS(nFingersToEnroll) |  
               UF_PACK_NTEMPLATES(nTemplatesPerFinger);  
  
// (2) excute the enrollment ui api  
UFS_VerifyUI(hScanner, m_nTimeout, UFS_TEMPLATE_TYPE_SUPREMA, nOptions,
```

```
pBuf, pBufSize, "verification.bmp", &nFingerIndex);
```

Return Values([refer to return values](#))

UFS_CaptureSingleUI

Performs same as UFS_CaptureSingle and Popup Window appears once the capturing starts to show captured image then disappears.

```
UFS_STATUS UFS_API UFS_CaptureSingleUI(  
    HUFScanner hScanner,  
    int nTimeout,  
    int nOptions,  
    BYTE* pUFIImageBuf,  
    int* pUFIImageWidth,  
    int* pUFIImageHeight,  
    char* pImages_Path ,  
    int* nFPQuality);
```

Parameters

- *hScanner [in]* : Effective handle for connected BioMini
- *nTimeout [in]* : Applicable timeout parameter for capture single function
- *nOptions [in]* : Same as UFS_EnrollUI option
- *pUFIImageBuf [in]* : Buffer of captured image (The memory buffer has to be managed by the user)
- *pUFIImageWidth, int* pUFIImageHeight [in]* :
- *pImages_Path [in]* : Width / Height of Captured image (pixel)
- *nFPQuality [in]* : Returns same score as UFS_GetFPQuality

Example

```
UFS_STATUS ufs_res = UFS_GetCaptureImageBufferInfo(hScanner ,&nWidth ,  
&nHeight, &nResolution );  
unsigned char *pbImage = (unsigned char*)malloc(nWidth * nHeight);  
ZeroMemory(pbImage , , nWidth* nHeight);  
int nOptions =  
    UF_PACK_SECURITY(hWnd->m_nSecurityLevel+1) |  
    UF_PACK_QUALITY(hWnd->m_quality);  
ufs_res = UFS_CaptureSingleUI(hScanner, hWnd->m_nTimeout, nOptions, pbImage,  
&nWidth , &nHeight, NULL, &nQuality);  
hWnd->AddMessage( "UFS_CaptureSingleUI FPQuality %s : %d\r\n",  
ufs_res==UFS_OK?"OK":"Failed", nQuality);  
free(pbImage);
```

Return Values[\(refer to return values\)](#)

UFM_Create

Creates a matcher object.

```
UFM_STATUS UFM_API UFM_Create(  
    HUFSMatcher* phMatcher );
```

Parameters

- *phMatcher [out]* : Pointer to handle of the matcher object

Examples

```
UFM_STATUS ufm_res;  
HUFSMatcher hMatcher;  
  
ufm_res = UFM_Create(&hMatcher);  
if (ufm_res == UFM_OK)  
{  
    // UFM_Create is succeeded  
}  
else  
{  
    // UFM_Create is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values [\(refer to return values\)](#)

UFM_Delete

Deletes specified matcher object.

```
UFM_STATUS UFM_API UFM_Delete(  
    HUFSMatcher pMatcher );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object

Examples

```
UFM_STATUS ufm_res;  
HUFSMatcher hMatcher;  
  
// Create hMatcher and use  
  
ufm_res = UFM_Delete(hMatcher);  
if (ufm_res == UFM_OK)  
{  
    // UFM_Delete is succeeded  
}  
else  
{  
    // UFM_Delete is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values [\(refer to return values\)](#)

UFM_GetParameter

Gets parameter value of UFMatcher module.

```
UFM_STATUS UFM_API UFM_GetParameter(  
    HUFMatcher pMatcher,  
    int nParam,  
    void* pValue );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFM_PARAM_FAST_MODE	301	Fast Mode (0: not use fast mode, 1: use fast mode)	1
UFM_PARAM_SECURITY_LEVEL	302	Set the False Accept Ratio(FAR) (1: Below 1%, 2: Below 0.1%, 3: Below 0.01%, 4: Below 0.001%, 5: Below 0.0001%, 6: Below 0.00001%, 7: Below 0.000001%)	4
UFM_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFM_PARAM_AUTO_ROTATE	321	Rotate Mode(0: not use rotate mode, 1: use rotate mode)	0
UFM_PARAM_SDK_VERSION	210	SDK Version (get only)	-
UFM_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	

- *pValue [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
int nValue;  
  
// Create hMatcher  
  
// Get fast mode  
ufm_res = UFM_GetParameter(hMatcher, UFM_PARAM_FAST_MODE, &nValue);  
// Error handling routine is omitted  
  
// Get security level  
ufm_res = UFM_GetParameter(hMatcher, UFM_PARAM_SECURITY_LEVEL, &nValue);  
// Error handling routine is omitted  
  
// Get use SIF  
ufm_res = UFM_GetParameter(hMatcher, UFM_PARAM_USE_SIF, &nValue);  
// Error handling routine is omitted
```

Return Values ([refer to return values](#))

UFM_SetParameter

Sets parameter value of UFMatcher module.

```
UFM_STATUS UFM_API UFM_SetParameter(  
    HUFMatcher pMatcher,  
    int nParam,  
    void* pValue );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFM_PARAM_FAST_MODE	301	Fast Mode (0: not use fast mode, 1: use fast mode)	1
UFM_PARAM_SECURITY_LEVEL	302	Set the False Accept Ratio(FAR) (1: Below 1%, 2: Below 0.1%, 3: Below 0.01%, 4: Below 0.001%, 5: Below 0.0001%, 6: Below 0.00001%, 7: Below 0.000001%)	4
UFM_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFM_PARAM_AUTO_ROTATE	321	Rotate Mode(0: not use rotate mode, 1: use rotate mode)	0
UFM_PARAM_SDK_VERSION	210	SDK Version (get only)	-
UFM_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	

- *pValue [in]* : Pointer to parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
int nValue;  
  
// Create hMatcher  
  
// Set fast mode to nValue  
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_FAST_MODE, &nValue);  
// Error handling routine is omitted  
  
// Set security level to nValue  
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_SECURITY_LEVEL, &nValue);  
// Error handling routine is omitted  
  
// Set use SIF to nValue  
ufm_res = UFM_SetParameter(hMatcher, UFM_PARAM_USE_SIF, &nValue);  
// Error handling routine is omitted
```

Return Values ([refer to return values](#))

UFM_Verify

Compares two extracted templates.

```
UFM_STATUS UFM_API UFM_Verify(  
    HUFMatcher pMatcher,  
    unsigned char* pTemplate1,  
    int nTemplate1Size,  
    unsigned char* pTemplate2,  
    int nTemplate2Size,  
    int* bVerifySucceed  
);
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *pTemplate1 [in]* : Pointer to the template1
- *nTemplate1Size [in]* : Specifies the size of the template1
- *pTemplate2 [in]* : Pointer to the template2
- *nTemplate2Size [in]* : Specifies the size of the template2
- *bVerifySucceed [out]* : Receives, whether verification is succeed; 1: verification is succeed, 0: verification is failed

Examples

```
// Assume template size is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
unsigned char Template1[MAX_TEMPLATE_SIZE];  
unsigned char Template2[MAX_TEMPLATE_SIZE];  
int nTemplate1Size;  
int nTemplate2Size;  
int bVerifySucceed;  
  
// Create hMatcher  
// Get two templates, Template1 and Template2  
  
ufm_res = UFM_Verify(hMatcher, Template1, nTemplate1Size, Template2,  
nTemplate2Size, &bVerifySucceed);  
if (ufm_res == UFM_OK)  
{  
    // UFM_Verify is succeeded  
    if (bVerifySucceed)  
    {  
        // Template1 is matched to Template2  
    }  
}
```

```
    else
    {
        // Template1 is not matched to Template2
    }
}
else
{
    // UFM_Verify is failed
    // Use UFM_GetErrorString function to show error string
}
```

Return Values ([refer to return values](#))

UFM_VerifyEx

Performs same as UFM_Verify, and returns matching score by 6th parameter (matching score in between 0~1, idle match as the score is close to 1)

```
UFM_STATUS UFM_API UFM_VerifyEx(  
    HUFMatcher hMatcher,  
    unsigned char* pTemplate1,  
    int nTemplate1Size,  
    unsigned char* pTemplate2,  
    int nTemplate2Size,  
    float* fScore,  
    int* bVerifySucceed  
);
```

Parameter Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pTemplate1 [in]* : Pointer to the template1
- *nTemplate1Size [in]* : Specifies the size of the template1
- *pTemplate2 [in]* : Pointer to the template2
- *nTemplate2Size [in]* : Specifies the size of the template2
- *fScore [out]* : Matching score between pTemplate1 and pTemplate2
- *bVerifySucceed [out]* : Receives, whether verification is succeed; 1: verification is succeed, 0: verification is failed

Example

```
// Assume template size is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
unsigned char Template1[MAX_TEMPLATE_SIZE];  
unsigned char Template2[MAX_TEMPLATE_SIZE];  
int nTemplate1Size;  
int nTemplate2Size;  
float fScore;  
int bVerifySucceed;  
  
// Create hMatcher  
// Get two templates, Template1 and Template2  
  
ufm_res = UFM_Verify(hMatcher, Template1, nTemplate1Size, Template2,  
nTemplate2Size, &fScore, &bVerifySucceed);  
if (ufm_res == UFM_OK)  
{  
    // UFM_Verify is succeeded  
    if (bVerifySucceed)
```

```
{
    // Template1 is matched to Template2
}
else
{
    // Template1 is not matched to Template2
}
}
else
{
    // UFM_Verify is failed
    // Use UFM_GetErrorString function to show error string
}
```

Return Values ([refer to return values](#))

UFM_Identify, UFM_IdentifyMT

Compares a template with given template array. UFM_IdentifyMT function uses multi threads internally for faster identifying in multi-core systems.

```
UFM_STATUS UFM_API UFM_Identify(  
    HUFMatcher pMatcher,  
    unsigned char* pTemplate1,  
    int nTemplate1Size,  
    unsigned char** ppTemplate2,  
    int* pnTemplate2Size,  
    int nTemplate2Num,  
    int nTimeout,  
    int* pnMatchTemplate2Index );  
  
UFM_STATUS UFM_API UFM_IdentifyMT(  
    HUFMatcher pMatcher,  
    unsigned char* pTemplate1,  
    int nTemplate1Size,  
    unsigned char** ppTemplate2,  
    int* pnTemplate2Size,  
    int nTemplate2Num,  
    int nTimeout,  
    int* pnMatchTemplate2Index );
```

Parameters

- *pMatcher [in]* : Handle of the matcher object
- *pTemplate1 [in]* : Pointer to the template
- *nTemplate1Size [in]* : Specifies the size of the template
- *ppTemplate2 [in]* : Pointer to the template array
- *pnTemplate2Size [in]* : Pointer to the template size array
- *nTemplate2Num [in]* : Specifies the number of templates in the template array
- *nTimeout [in]* : Specifies maximum time for identifying in milliseconds; If elapsed time for identifying exceeds nTimeout, function stops further identifying and returns UFM_ERR_MATCH_TIMEOUT; 0 means infinity
- *pnMatchTemplate2Index [out]* : Receives the index of matched template in the template array; -1 means pTemplate1 is not matched to all of templates in ppTemplate2

Examples

```
// Assume template size is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024 UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
unsigned char Template1[MAX_TEMPLATE_SIZE];  
unsigned char** ppTemplate2;  
int nTemplate1Size;  
int* pnTemplate2Size;
```

```

int nTemplate2Num;
int nTimeout;
int nMatchTemplate2Index;
int i;

// Create hMatcher

// Get input template from user, Template1

// Make template array from DB or something
// Get number of template to nTemplate2Num
ppTemplate2 = (unsigned char**)malloc(nTemplate2Num * sizeof(unsigned
char*));
pnTemplate2Size = (int*)malloc(nTemplate2Num * sizeof(int));
for (i = ; i < nTemplate2Num; i++)
{
    ppTemplate2[i] = (unsigned char*)malloc(MAX_TEMPLATE_SIZE *
sizeof(unsigned char));
    // Copy i th template to ppTemplate2[i]
    // Set i th template size to pnTemplateSize[i]
}

// Set match timeout to nTimeout

ufm_res = UFM_Identify(hMatcher, Template1, Template1Size, ppTemplate2,
pnTemplate2Size, nTemplate2Num, nTimeout, &nMatchTemplate2Index);

if (ufm_res == UFM_OK)
{
    // UFM_Identify is succeeded
    if (nMatchTemplate2Index != -1)
    {
        // Input fingerprint Template1 is matched to
ppTemplate2[nMatchTemplate2Index]
    }
    else
    {
        // Input fingerprint is not in ppTemplate2
    }
}
else
{
    // UFM_Identify is failed
    // Use UFM_GetErrorString function to show error string
}

// Free template array
free(pnTemplate2Size);
for (i = ; i < nTemplate2Num; i++)
{
    free(ppTemplate2[i]);
}

```

```
}  
  
free(ppTemplate2);
```

Return Values ([refer to return values](#))

UFM_AbortIdentify

Aborts current identifying procedure started using [UFM_Identify](#).

```
UFM_STATUS UFM_API UFM_AbortIdentify(  
    HUFSMatcher pMatcher );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object

Examples

```
UFM_STATUS ufm_res;  
HUFSMatcher hMatcher;  
  
// Create hMatcher  
// Start UFM_Identify  
  
ufm_res = UFM_AbortIdentify(hMatcher);  
if (ufm_res == UFM_OK)  
{  
    // UFM_AbortIdentify is succeeded  
}  
else  
{  
    // UFM_AbortIdentify is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values ([refer to return values](#))

UFM_IdentifyInit

Initializes identify with input template.

```
UFM_STATUS UFM_API UFM_IdentifyInit(  
    HUFMatcher pMatcher,  
    unsigned char* pTemplate1,  
    int nTemplate1Size, );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *pTemplate1 [in]* : Pointer to the template
- *nTemplate1Size [in]* : Specifies the size of the template

Examples

```
// Assume template size is 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
unsigned char Template1[MAX_TEMPLATE_SIZE];  
int nTemplate1Size;  
  
// Create hMatcher  
// Get Template1  
  
ufm_res = UFM_IdentifyInit(hMatcher, Template1, nTemplate1Size);  
if (ufm_res == UFM_OK)  
{  
    // UFM_IdentifyInit is succeeded  
}  
else  
{  
    // UFM_IdentifyInit is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values [\(refer to return values\)](#)

UFM_IdentifyNext

Matches one input template to the template specified in [UFM_IdentifyInit](#).

```
UFM_STATUS UFM_API UFM_IdentifyNext(  
    HUFLMatcher pMatcher,  
    unsigned char* pTemplate2,  
    int nTemplate2Size,  
    int* bIdentifySucceed );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *pTemplate2 [in]* : Pointer to the template array
- *nTemplate2Size [in]* : Specifies the size of the template array
- *bIdentifySucceed [out]* : Receives whether identification is succeed; 1: identification is succeed, 0: identification is failed

Examples

```
#define MAX_TEMPLATE_NUM 50  
  
UFM_STATUS ufm_res;  
HUFLMatcher hMatcher;  
unsigned char *Template2[MAX_TEMPLATE_NUM * 2];  
int nTemplate2Size[MAX_TEMPLATE_NUM * 2];  
int nTemplate2Num;  
int bIdentifySucceed;  
int i;  
  
// Create hMatcher  
// Execute UFM_IdentifyInit with query template  
// Get number of templates in DB or something, and save it to nTemplate2Num  
  
bIdentifySucceed = ;  
for (i = ; i < nTemplate2Num; i++)  
{  
    // Get one template in DB or something, and save it to Template2 and  
    nTemplate2Size  
  
    ufm_res = UFM_IdentifyNext(hMatcher, Template2[i], nTemplate2Size[i],  
&bIdentifySucceed);  
    if (ufm_res == UFM_OK)  
    {  
        // UFM_IdentifyNext is succeeded  
    }  
    else  
    {  

```

```
    // UFM_IdentifyNext is failed
    // Use UFM_GetErrorString function to show error string
    // return;
}
if (bIdentifySucceed)
{
    // Identification is succeed break;
}
}
if (!bIdentifySucceed)
{
    // Identification is failed
}
```

Return Values [\(refer to return values\)](#)

UFM_RotateTemplate

Rotates the specified template to the amount of 180 degrees.

```
UFM_STATUS UFM_API UFM_RotateTemplate(  
    HUFMatcher pMatcher,  
    unsigned char* pTemplate,  
    int nTemplateSize );
```

Parameters

- *pMatcher[in]* : Handle to the matcher object
- *pTemplate [in / out]* : Pointer to the template
- *nTemplateSize [in]* : Specifies the size of the template

Examples

```
// Assume template size if 1024 bytes  
#define MAX_TEMPLATE_SIZE 1024  
  
UFM_STATUS ufm_res;  
HUFMatcher hMatcher;  
unsigned char Template[MAX_TEMPLATE_SIZE];  
int nTemplateSize;  
  
// Create hMatcher  
// Get a template, and save it to Template and nTemplateSize  
  
ufm_res = UFM_RotateTemplate(hMatcher, Template, nTemplateSize);  
if (ufm_res == UFM_OK)  
{  
    // UFM_RotateTemplate is succeeded  
}  
else  
{  
    // UFM_RotateTemplate is failed // Use UFM_GetErrorString function to  
    show error string  
}
```

Return Values [\(refer to return values\)](#)

UFM_GetErrorString

Gets the error string for specified [UFM_STATUS](#) value.

```
UFM_STATUS UFM_API UFM_GetErrorString(  
    UFM_STATUS res,  
    char* szErrorString );
```

Parameters

- *res [in]* : Status return value
- *szErrorString [out]* : Receives error string

Examples

```
UFM_STATUS ufm_res;  
char strError[128];  
  
// Get status return code,  
  
ufm_res ufm_res = UFM_GetErrorString(ufm_res, strError);  
if (ufm_res == UFM_OK)  
{  
    // UFM_GetErrorString is succeeded  
}  
else  
{  
    // UFM_GetErrorString is failed  
}
```

Return Values ([refer to return values](#))

UFM_GetTemplateType

Gets the parameter value.

```
UFM_STATUS UFM_API UFM_GetTemplateType(  
    HUFTMatcher pMatcher,  
    int* nTemplateType );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *nTemplateType [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to template type

Template type	Code	Description
UFM_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFM_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFM_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
UFM_STATUS ufm_res;  
HUFTMatcher hMatcher;  
int nTemplateType;  
  
// Get hMatcher handle  
  
ufm_res = UFM_GetTemplateType(hMatcher,&nTemplateType);  
// Error handling routine is omitted
```

Return Values [\(refer to return values\)](#)

UFM_SetTemplateType

Gets parameter value.

```
UFM_STATUS UFM_API UFM_SetTemplateType(  
    HUFTMatcher pMatcher,  
    int nTemplateType );
```

Parameters

- *pMatcher [in]* : Handle to the matcher object
- *nTemplateType [in]* : Parameter type; one of template type

Template type	Code	Description
UFM_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFM_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFM_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
UFM_STATUS ufm_res;  
HUFTMatcher hMatcher;  
int nTemplateType;  
  
// Get hScanner handle  
  
nTemplateType = UFM_TEMPLATE_TYPE_SUPREMA;  
  
ufs_res = UFM_SetTemplateType(hMatcher, nTemplateType);  
// Error handling routine is omitted
```

Return Values [\(refer to return values\)](#)

UFS_STATUS

Every function in a UFSscanner module returns **UFS_OK** when it succeeds. When it fails, it returns a value corresponding to a error code. Please find the error code on the followings if you'd like to know the information about the UFS_STATUS (integer) value.

Code	Value	Description
UFS_OK	0	Success
UFS_ERROR	-1	General error
UFS_ERR_NO_LICENSE	-101	Device is not connected or License is not located
UFS_ERR_LICENSE_NOT_MATCH	-102	License does not match
UFS_ERR_LICENSE_EXPIRED	-103	License has expired
UFS_ERR_NOT_SUPPORTED	-111	This function is not supported
UFS_ERR_INVALID_PARAMETERS	-112	Input parameters are invalid
UFS_ERR_ALREADY_INITIALIZED	-201	Module is already initialized
UFS_ERR_NOT_INITIALIZED	-202	Module is not initialized
UFS_ERR_DEVICE_NUMBER_EXCEED	-203	Device number exceeds
UFS_ERR_LOAD_SCANNER_LIBRARY	-204	Error on loading the library of a scanner
UFS_ERR_CAPTURE_RUNNING	-211	Capturing is started using UFS_CaptureSingleImage or UFS_StartCapturing
UFS_ERR_CAPTURE_FAILED	-212	Capturing is timeout or aborted
UFS_ERR_NOT_GOOD_IMAGE	-301	Input image is not good
UFS_ERR_EXTRACTION_FAILED	-302	Extraction is failed
UFS_ERR_CORE_NOT_DETECTED	-351	Core is not detected
UFS_ERR_CORE_TO_LEFT_TOP	-353	Move finger to left-top
UFS_ERR_CORE_TO_TOP	-354	Move finger to top
UFS_ERR_CORE_TO_RIGHT_TOP	-355	Move finger to right-top
UFS_ERR_CORE_TO_RIGHT	-356	Move finger to right
UFS_ERR_CORE_TO_RIGHT_BOTTOM	-357	Move finger to right-bottom
UFS_ERR_CORE_TO_BOTTOM	-358	Move finger to bottom
UFS_ERR_CORE_TO_LEFT_BOTTOM	-359	Move finger to left-bottom

UFM_STATUS

Every function in a UFMatcher module returns **UFM_OK** when it succeeds. When it fails, it returns a value corresponding to a error code. Please find the error code on the followings if you'd like to know the information about the UFM_STATUS (integer) value.

Code	Value	Description
UFM_OK	0	Success
UFM_ERROR	-1	General error
UFM_ERR_NO_LICENSE	-101	System has no license
UFM_ERR_LICENSE_NOT_MATCH	-102	License does not match
UFM_ERR_LICENSE_EXPIRED	-103	License has expired
UFM_ERR_NOT_SUPPORTED	-111	This function is not supported
UFM_ERR_INVALID_PARAMETERS	-112	Input parameters are invalid
UFM_ERR_MATCH_TIMEOUT	-401	Matching is timeout
UFM_ERR_MATCH_ABORTED	-402	Matching is aborted
UFM_ERR_TEMPLATE_TYPE	-411	Template type does not match

4. Java Development

4.1 Java Installation guide

0. Required product

You need the following packages installed

Image SDK, Java SDK 1.4 or later, JNI package file (ImageSDK.jar is located at "<image sdk installed path>/bin/java")

1. Install Java SDK

You must install Java SDK 1.4. or later

Please see <http://www.oracle.com> site.

Copy the jna.jar to "<java sdk installed path>/lib"

2.Set Classpath

After java sdk installation,you must set classpath as following example.

"CLASSPATH=.:<path to the ImageSDK.jar file >:"

3.Build and run sample

You can build and run the JNI demo application in the following location.

<installed folder>/samples/java/demoImageSDK.java

4.2 Java APIs



The Java API is built as a wrapper to the JNI. The Java API is available for Linux. This page provides an overview of the API. For details of using the API on a specific reader platform, consult the appropriate Platform Guide.

[UFS_Init\(\)](#)

Initializes a UFScanner module

[UFS_Update\(\)](#)

Enforces a UFScanner module to update the connection state of scanners

[UFS_Uninit\(\)](#)

Un-initializes a UFScanner module

[UFS_SetScannerCallback\(\)](#)

Registers a scanner callback function

[UFS_RemoveScannerCallback\(\)](#)

Removes a registered scanner callback function

[UFS_GetScannerNumber\(\)](#)

Gets the number of scanners

[UFS_GetScannerHandle\(\)](#)

Gets the scanner handle using the scanner index

[UFS_GetScannerHandleByID\(\)](#)

Gets the scanner handle using a scanner ID

[UFS_GetScannerIndex\(\)](#)

Gets the scanner index that is assigned to the scanner handle

[UFS_GetScannerID\(\)](#)

Gets a scanner ID that is assigned to the scanner handle

[UFS_GetScannerType\(\)](#)

Gets the scanner type that is assigned to the scanner handle

[UFS_GetParameter\(\)](#)

Gets the parameter value of a UFScanner module

[UFS_SetParameter\(\)](#)

Sets the parameter value of a UFScanner module

UFS_IsSensorOn()

Checks whether a scanner is connected or not

UFS_IsFingerOn()

Checks whether a finger is placed on a scanner or not

UFS_CaptureSingleImage()

Captures single image. Captured image is stored to the internal buffer

UFS_StartCapturing()

Starts capturing. The capture is going on until the specified time exceeds

UFS_StartAutoCapture()

Starts the automatic capture. Currently this function is working for Suprema SFR600(BioMini Slim) only

UFS_IsCapturing()

Checks if the specified scanner is running to capture which is started by UFS_CaptureSingleImage or UFS_StartCapturing

UFS_AbortCapturing()

Aborts capturing which is started by UFS_CaptureSingleImage() or UFS_StartCapturing()

UFS_Extract()

Extracts a template from the stored image buffer which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing()

UFS_ExtractEx()

Extracts a template from the stored image buffer which is acquired using UFS_CaptureSingleImage() or UFS_StartCapturing(). This is extended version of UFS_Extract() function to accommodate a template with large size

UFS_SetEncryptionKey()

Sets the encryption key

UFS_EncryptTemplate()

Encrypts a template

UFS_DecryptTemplate()

Decrypt a template

UFS_GetCaptureImageBufferInfo()

Gets the information of the capture image buffer

UFS_GetCaptureImageBuffer()

Copies the capture image buffer to the specified image data array

UFS_GetCaptureImageBufferToBMPIImageBuffer()

Copies the capture image buffer to the specified image data of bmp format

UFS_GetCaptureImageBufferTo19794_4ImageBuffer()

Copies the capture image buffer to the specified image data of 19794_4 format

[UFS_GetCaptureImageBufferToWSQImageBuffer\(\)](#)

Copies the capture image buffer to the specified image data of the WSQ format

[UFS_GetCaptureImageBufferToWSQImageBufferVar\(\)](#)

Copies the capture image buffer (cropped or expanded to the specified size) to the target image data buffer of the WSQ format

[UFS-DecompressWSQBMP\(\)](#)

Decompress a WSQ file and save to a BMP file

[UFS-DecompressWSQBMPMem\(\)](#)

Decompress a WSQ buffer and save to the image data of the bmp format

[UFS_SaveCaptureImageBufferToBMP\(\)](#)

Saves the capture image buffer to the specified file of the bitmap format

[UFS_SaveCaptureImageBufferTo19794_4\(\)](#)

Saves the capture image buffer to the specified file of the 19794_4 format

[UFS_SaveCaptureImageBufferToWSQ\(\)](#)

Saves the capture image buffer to the specified file of the WSQ format

[UFS_SaveCaptureImageBufferToWSQVar\(\)](#)

Saves the capture image buffer (cropped or expanded to the specified size) to the target file of the WSQ format

[UFS_ClearCaptureImageBuffer\(\)](#)

Clears the capture image buffer stored to the internal buffer

[UFS_GetErrorString\(\)](#)

Gets the error string for specified UFS_STAUS value

[UFS_GetTemplateType\(\)](#)

Gets the template type value

[UFS_SetTemplateType\(\)](#)

Sets the template type value

[UFS_SelectTemplate\(\)](#)

Selects n number of good templates from m number of input templates

[UFS_SelectTemplateEx\(\)](#)

Selects n number of good templates from m number of input templates This is extended version of UFS_SelectTemplate function to accommodate the template with large size

[UFS_GetFPQuality\(\)](#)

Calculates the quality score of an image as defined in NISTIR 7151: FingerPrint Image Quality. The score would be between 1(excellent) and 5(poor)

UFS_GetFeatureNumber()

Get number of Minutiae from template data

UFM_Create()

Creates a matcher object

UFM_Delete()

Deletes a specified matcher object

UFM_GetParameter()

Gets the parameter value of UFMatcher module

UFM_SetParameter()

Sets the parameter value of UFMatcher module

UFM_Verify()

Compares two extracted templates

UFM_Identify(),IdentifyMT()

Compares a template with given template array UFM_IdentifyMT() function uses multi threads internally for faster identifying in multi-core systems

UFM_AbortIdentify()

Aborts current identifying procedure started using UFM_Identify()

UFM_IdentifyInit()

Initializes identify with input template

UFM_IdentifyNext()

Matches one input template to the template specified in UFM_IdentifyInit()

UFM_RotateTemplate()

Rotates the specified template to the amount of 180 degrees

UFM_GetErrorString()

Gets the error string for the specified UFM_STAUS value

UFM_GetTemplateType()

Gets the parameter value

UFM_SetTemplateType()

Sets the parameter value

UFS_Init

Initializes a UFScanner module.

```
int UFS_Init();
```

Examples

```
int nRes =;
BioMiniSDK p = null;

//make instance
p = new BioMiniSDK();

nRes = p.UFS_Init();

if(nRes ==p.UFS_OK)
{
    // UFS_Init is succeeded
}
else
{
    // UFS_Init is failed
    // Use UFS_GetErrorString method to show error string
}
```

Return Values([refer to return values](#))

UFS_Update

Enforces a UFSscanner module to update the connection state of scanners.

```
int UFS_Update();
```

Examples

```
int ufs_res;

//make class library instance(BioMiniSDK p)

ufs_res = p.UFS_Update();
if (ufs_res == p.UFS_OK)
{
    // UFS_Update is succeeded
}
else
{
    // UFS_Update is failed
    // Use UFS_GetErrorString function to show error string
}
```

Return Values([refer to return values](#))

UFS_Uninit

Un-initializes a UFSscanner module.

```
int UFS_UnInit();
```

Examples

```
int ufs_res;  
  
//make class library instance(BioMiniSDK p)  
  
ufs_res = p.UFS_Uninit();  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_Uninit is succeeded  
}  
else  
{  
    // UFS_Uninit is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_SetScannerCallback

Registers the scanner callback function.

```
int UFS_SetScannerCallback(  
    String nCallbackFunctionName);
```

Parameters

- *nCallbackFunctionName[in]* :

Examples

```
// Define scanner procedure  
  
public void scannerCallback(char[] szScannerID, int bSensorOn)  
{  
    // ...  
}  
  
// Set parameter, the name of call function for scanner event (USB Plug)  
//make class library instance(BioMiniSDK p)  
  
ufs_res = p.UFS_SetScannerCallback("scannerCallback");  
if(ufs_res==p.UFS_OK)  
{  
    // UFS_SetScannerCallback is succeeded  
}  
else  
{  
    // UFS_SetScannerCallback is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_RemoveScannerCallback

Removes the registered scanner callback function.

```
int UFS_RemoveScannerCallback();
```

Examples

```
int ufs_res;  
  
//make class library instance(BioMiniSDK p)  
ufs_res = p.UFS_RemoveScannerCallback();  
if (ufs_res == UFS_OK)  
{  
    // UFS_RemoveScannerCallback is succeeded  
}  
else  
{  
    // UFS_RemoveScannerCallback is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_GetScannerNumber

Gets the number of scanners.

```
int UFS_GetScannerNumber(  
    int[] pnScannerNumber );
```

Parameters

- *pnScannerNumber [out]* : Receive the number of scanners

Examples

```
int ufs_res;  
int[] nNumber = new int[1];  
  
//make class library instance(BioMiniSDK p)  
  
ufs_res = p.UFS_GetScannerNumber(nNumber);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerNumber is succeeded  
    nNumber[1] ...  
}  
else  
{  
    // UFS_GetScannerNumber is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerHandle

Gets the scanner handle using a scanner index.

```
int UFS_GetScannerHandle(  
    int nScannerIndex,  
    long[] phScanner );
```

Parameters

- *nScannerIndex [in]* : Scanner index (0 ~ number of scanners - 1)
- *phScanner [out]* : Pointer to handle of the scanner object

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int index = ;  
  
// Set nScannerIndex to (0 ~ number of scanners - 1 )  
// Number of scanner can be retrieved using UFS_GetScannerNumber function  
//make class library instance(BioMiniSDK p)  
  
ufs_res = p.UFS_GetScannerHandle(index, hScanner);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerHandle is succeeded  
    // hScanner[0] is the handle  
}  
else  
{  
    // UFS_GetScannerHandle is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerHandleByID

Gets the scanner handle using a scanner ID.

```
int UFS_GetScannerHandleByID(  
    String szScannerID,  
    long[] phScanner );
```

Parameters

- *szScannerID [in]* : Scanner ID
- *phScanner [out]* : Pointer to handle of the scanner object

Return Values([refer to return values](#))

UFS_GetScannerIndex

Gets a scanner index that is assigned to the scanner handle.

```
int UFS_GetScannerIndex(  
    long hScanner,  
    int[] pnScannerIndex );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnScannerIndex [out]* : Receive scanner index of specified scanner handle

Return Values([refer to return values](#))

UFS_GetScannerID

Gets scanner ID assigned to scanner handle.

```
int UFS_GetScannerID(  
    long hScanner,  
    byte[] szScannerID );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szScannerID [out]* : Receive scanner ID of specified scanner handle; Scanner ID has maximum 32 characters. szScannerID must be allocated in user's applications and allocated size must be larger than 33 bytes for considering null character in 33th byte position.

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
byte[] strID = new byte[128];  
  
    // Should be larger than 33 bytes  
    // make class library instance(BioMiniSDK p)  
    // Get hScanner handle  
  
ufs_res = p.UFS_GetScannerID(hScanner[], strID);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_GetScannerID is succeeded  
    // byte to string..  
}  
else  
{  
    // UFS_GetScannerID is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetScannerType

Gets the scanner type that is assigned to the scanner handle.

```
int UFS_GetScannerType(  
    long hScanner,  
    int[] pnScannerType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnScannerType [out]* : Receives one of the scanner type

Scanner type	Code	Description
UFS_SCANNER_TYPE_SFR200	1001	Suprema SFR200
UFS_SCANNER_TYPE_SFR300	1002	Suprema SFR300-S
UFS_SCANNER_TYPE_SFR300v2	1003	Suprema SFR300v2, SFR400
UFS_SCANNER_TYPE_SFR500	1004	Suprema SFR500
UFS_SCANNER_TYPE_SFR600	1005	Suprema SFR600
UFS_SCANNER_TYPE_SFR410	1006	Suprema SFR410
UFS_SCANNER_TYPE_SFR550	1007	Suprema SFR550

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int[] nScannerType = new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_GetScannerType(hScanner[], nScannerType);  
if (ufs_res == UFS_OK)  
{  
    // UFS_GetScannerType is succeeded  
    // nScannerType[0] is the scanner type  
}  
else  
{  
    // UFS_GetScannerType is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_GetParameter

Gets parameter value of UFSscanner module.

```
int UFS_GetParameter(  
    long hScanner,  
    int nParam,  
    int[] pValue );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFS_PARAM_TIMEOUT	201	Timeout (millisecond unit) (0: infinite)	5000
UFS_PARAM_BRIGHTNESS	202	Brightness (0 ~ 255); Higher value means darker image. * Not supported Device: BioMini-Slim(SFU-S20, SFU-S20B) BioMini(SFU-300)	100
UFS_PARAM_SENSITIVITY	203	Sensitivity (0 ~ 7); Higher value means more sensitive	4
UFS_PARAM_SERIAL	204	Serial (get only)	-
UFS_PARAM_SDK_VERSION	210	SDK Version (get only)	
UFS_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	
UFS_PARAM_DETECT_CORE	301	Detect core (0: not use core, 1: use core)	0
UFS_PARAM_TEMPLATE_SIZE	302	Template size (byte unit) (256 ~ 1024, 32 bytes step size)	1024
UFS_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFS_PARAM_DETECT_FAKE	312	Use live Finger Detection (0: not use LFD, 1 ~ 3 : use LFD); Higher value means more strong to fake finger * Supported Device: BioMini Slim(SFU-S20, SFU-S20B)	0

- *pValue [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int[] nValue = new int[1];  
  
//make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Get timeout  
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_TIMEOUT, nValue );  
// Error handling routine is omitted  
// nValue[0] is the parameter value  
  
// Get brightness
```

```
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_BRIGHTNESS, nValue );  
// Error handling routine is omitted  
  
// Get sensitivity  
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_SENSITIVITY, nValue );  
// Error handling routine is omitted  
  
// Get detect core  
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_DETECT_CORE, nValue );  
// Error handling routine is omitted  
  
// Get template size  
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_TEMPLATE_SIZE, nValue  
);  
// Error handling routine is omitted  
  
// Get use SIF  
ufs_res = p.UFS_GetParameter(hScanner[], p.UFS_PARAM_USE_SIF, nValue );  
// Error handling routine is omitted
```

Return Values(refer to return values)

UFS_SetParameter

Sets parameter value of UFSscanner module.

```
int UFS_SetParameter(  
    long hScanner,  
    int nParam,  
    int[] pValue );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFS_PARAM_TIMEOUT	201	Timeout (millisecond unit) (0: infinite)	5000
UFS_PARAM_BRIGHTNESS	202	Brightness (0 ~ 255); Higher value means darker image. * Not supported Device: BioMini-Slim(SFU-S20, SFU-S20B) BioMini(SFU-300)	100
UFS_PARAM_SENSITIVITY	203	Sensitivity (0 ~ 7); Higher value means more sensitive	4
UFS_PARAM_SERIAL	204	Serial (get only)	-
UFS_PARAM_SDK_VERSION	210	SDK Version (get only)	
UFS_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	
UFS_PARAM_DETECT_CORE	301	Detect core (0: not use core, 1: use core)	0
UFS_PARAM_TEMPLATE_SIZE	302	Template size (byte unit) (256 ~ 1024, 32 bytes step size)	1024
UFS_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFS_PARAM_DETECT_FAKE	312	Use live Finger Detection (0: not use LFD, 1 ~ 3 : use LFD); Higher value means more strong to fake finger * Supported Device: BioMini Slim(SFU-S20, SFU-S20B)	0

- *pValue [in]* : Pointer to parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int[] nValue = new int[1];  
  
//make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Set timeout to nValue nValue[0] = 5000;  
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_TIMEOUT, nValue);  
// Error handling routine is omitted  
  
// Set brightness to nValue nValue[0] = 100;  
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_BRIGHTNESS, nValue);
```



```
// Error handling routine is omitted

// Set sensitivity to nValue
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_SENSITIVITY, nValue);
// Error handling routine is omitted

// Set detect core to nValue
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_DETECT_CORE, nValue);
// Error handling routine is omitted

// Set template size to nValue
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_TEMPLATE_SIZE, nValue);
// Error handling routine is omitted

// Set use SIF to nValue
ufs_res = p.UFS_SetParameter(hScanner[], p.UFS_PARAM_USE_SIF, nValue);
// Error handling routine is omitted
```

Return Values([refer to return values](#))

UFS_IsSensorOn

Checks whether a scanner is connected or not.

```
int UFS_IsSensorOn(  
    long hScanner,  
    int[] pbSensorOn );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pbSensorOn [out]* : Receive the status of specified scanner object; 1: the scanner is connected, 0: the scanner is disconnected

Return Values([refer to return values](#))

UFS_IsFingerOn

Checks whether a finger is placed on a scanner or not.

```
int UFS_IsFingerOn(  
    long hScanner,  
    int[] pbFingerOn );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pbFingerOn [out]* : Checks if a finger is placed on the specified scanner; 1: a finger is on the scanner, 0: a finger is not on the scanner

Return Values([refer to return values](#))

UFS_CaptureSingleImage

Captures single image. Captured image is stored to the internal buffer.

```
int UFS_CaptureSingleImage(  
    long hScanner);
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_CaptureSingleImage(hScanner[]);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_CaptureSingleImage is succeeded  
}  
else  
{  
    // UFS_CaptureSingleImage is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_StartCapturing

Starts capturing. The capture is going on until the specified time exceeds.

```
int UFS_StartCapturing(  
    long hScanner,  
    String nCallbackFunctionName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nCallbackFunctionName [in]* :

Examples

```
// Define capture procedure  
  
public void captureCallback(int bFingerOn, byte[] pImage, int nWidth, int  
nHeight, int nResolution)  
{  
    // ....  
    // pMainInstance.drawCurrentFingerImage();  
}  
  
int ufs_res;  
long[] hScanner = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Set parameter, the name of your call function for getting captured image  
ufs_res = p.UFS_StartCapturing(hScanner[], "captureCallback");  
  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_StartCapturing is succeeded  
}  
else  
{  
    // UFS_StartCapturing is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_StartAutoCapture

Starts the automatic capture. Currently this function is working for Suprema SFR600(BioMini Slim) only.

```
int UFS_StartAutoCapture(  
    long hScanner,  
    String nCallbackFunctionName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nCallbackFunctionName [in]* :

Examples

```
// Define capture procedure  
  
public void captureCallback(int bFingerOn, byte[] pImage, int nWidth, int  
nHeight, int nResolution)  
{  
    // ....  
    // pMainInstance.drawCurrentFingerImage();  
}  
  
int ufs_res;  
long[] hScanner = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Set parameter, the name of your call function for getting captured image  
  
ufs_res = p.UFS_StartAutoCapture(hScanner[], "captureCallback");  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_StartAutoCapture is succeeded  
}  
else  
{  
    // UFS_StartAutoCapture is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_IsCapturing

Checks if the specified scanner is running to capture which is started by [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#)

```
int UFS_IsCapturing(  
    long hScanner,  
    int[] bCapturing );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *bCapturing [out]* : Checks if the specified scanner is running capturing; 1: the capture is running, 0: the capture is not running

Return Values([refer to return values](#))

UFS_AbortCapturing

Aborts capturing which is started by [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#).

```
int UFS_AbortCapturing(  
    long hScanner);
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_AbortCapturing(hScanner[0]);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_AbortCapturing is succeeded  
}  
else  
{  
    // UFS_AbortCapturing is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_Extract

Extracts a template from the stored image buffer which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#).

```
int UFS_Extract(  
    long hScanner,  
    byte[] pTemplate,  
    int[] pnTemplateSize,  
    int[] pnEnrollQuality );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplate [out]* : Pointer to the template array; The array must be allocated in advance
- *pnTemplateSize [out]* : Receives the size (in bytes) of pTemplate
- *pnEnrollQuality [out]* : Receives the quality of enrollment; Quality value ranges from 1 to 100. Typically this value should be above 30 for further processing such as enroll and matching. Especially in case of enrollment, the use of good quality image (above 50) is highly recommended.

Return Values[\(refer to return values\)](#)

UFS_ExtractEx

Extracts a template from the stored image buffer which is acquired using [UFS_CaptureSingleImage](#) or [UFS_StartCapturing](#). This is extended version of [UFS_Extract](#) function to accommodate large size template.

```
int UFS_ExtractEx(  
    long hScanner,  
    int nBufferSize,  
    byte[] pTemplate,  
    int[] pnTemplateSize,  
    int[] pnEnrollQuality );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nBufferSize [in]* : Template buffer size
- *pTemplate [out]* : Pointer to the template array; The array must be allocated in advance
- *pnTemplateSize [out]* : Receives the size (in bytes) of pTemplate
- *pnEnrollQuality [out]* : Receives the quality of enrollment; Quality value ranges from 1 to 100. Typically this value should be above 30 for further processing such as enroll and matching. Especially in case of enrollment, the use of good quality image (above 50) is highly recommended.

Examples

```
// Template size can be controlled by using UFS_SetParameter function  
// Default value is 1024 bytes  
int MAX_TEMPLATE_SIZE = 1024;  
  
int ufs_res;  
long[] hScanner = new long[1];  
byte[] Template = new byte[MAX_TEMPLATE_SIZE];  
int[] TemplateSize= new int[1];  
int[] TemplateQuality= new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_ExtractEx(hScanner[], MAX_TEMPLATE_SIZE, Template,  
TemplateSize, TemplateQuality);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_ExtractEx is succeeded  
}  
else  
{
```

```
// UFS_ExtractEx is failed  
// Use UFS_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFS_SetEncryptionKey

Sets encryption key.

```
int UFS_SetEncryptionKey(  
    long hScanner,  
    byte[] pKey );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pKey[out]* : Pointer to the 32 bytes key array; default key is first byte is 1 and second to 32th byte are all 0

Return Values([refer to return values](#))

UFS_EncryptTemplate

Encrypts template.

```
int UFS_EncryptTemplate(  
    long hScanner,  
    byte[] pTemplateInput,  
    int nTemplateInputSize,  
    byte[] pTemplateOutput,  
    int[] pnTemplateOutputSize );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplateInput [in]* : Pointer to input template data
- *nTemplateInputSize [in]* : Input template size
- *pTemplateOutput [out]* : Pointer to encrypted template data
- *pnTemplateOutputSize [in / out]* : Inputs allocated size of encrypted template data; Receives output template size

Return Values([refer to return values](#))

UFS_DecryptTemplate

Decrypts template.

```
int UFS_DecryptTemplate(  
    long hScanner,  
    byte[] pTemplateInput,  
    int nTemplateInputSize,  
    byte[] pTemplateOutput,  
    int[] pnTemplateOutputSize );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplateInput [in]* : Pointer to input template data(encrypted)
- *nTemplateInputSize [in]* : Input template size
- *pTemplateOutput [out]* : Pointer to output template data
- *pnTemplateOutputSize [in / out]* : Inputs allocated size of output template data; Receives output template size

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferInfo

Gets the information of the capture image buffer.

```
int UFS_GetCaptureImageBufferInfo(  
    long hScanner,  
    int[] pnWidth,  
    int[] pnHeight,  
    int[] pnResolution );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pnWidth [out]* : Receives the width of the capture image buffer
- *pnHeight [out]* : Receives the height of the capture image buffer
- *pnResolution [out]* : Receives the resolution of the capture image buffer

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int[] nWidth = new int[1];  
int[] nHeight = new int[1];  
int[] nResolution = new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_GetCaptureImageBufferInfo(hScanner[], nWidth, nHeight,  
nResolution);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_GetCaptureImageBufferInfo is succeeded  
}  
else  
{  
    // UFS_GetCaptureImageBufferInfo is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetCaptureImageBuffer

Copies the capture image buffer to the specified image data array.

```
int UFS_GetCaptureImageBuffer(  
    long hScanner,  
    byte[] pImageData );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to image data array; The array must be allocated bigger than the size of capture image buffer in advance

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int[] nWidth = new int[1];  
int[] nHeight= new int[1];  
int[] nResolution= new int[1];  
  
// Get hScanner handle  
  
// Get capture image buffer information  
ufs_res = p.UFS_GetCaptureImageBufferInfo(hScanner[], nWidth, nHeight,  
nResolution);  
// Error handling routine is omitted  
  
// Allocate image buffer  
byte[] pImageData = new byte[nWidth * nHeight];  
  
ufs_res = p.UFS_GetCaptureImageBuffer(hScanner[], pImageData);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_GetCaptureImageBuffer is succeeded  
}  
else  
{  
    // UFS_GetCaptureImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFS_GetCaptureImageBufferToBMPImageBuffer

Copies the capture image buffer to the specified image data of bmp format.

```
int UFS_GetCaptureImageBufferToBMPImageBuffer(  
    long hScanner,  
    byte[] pImageData,  
    int[] pImageLength );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to bmp image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *pImageLength [out]* : pointer to bmp image data size

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferTo19794_4ImageBuffer

Copies the capture image buffer to the specified image data of 19794_4 format.

```
int UFS_GetCaptureImageBufferTo19794_4ImageBuffer(  
    long hScanner,  
    byte[] pImageData,  
    int[] pImageLength );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pImageData [out]* : Pointer to 19794_4 format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *pImageLength [out]* : pointer to 19794_4 format image data size

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferToWSQImageBuffer

Copies the capture image buffer to the specified image data of the WSQ format.

```
int UFS_GetCaptureImageBufferToWSQImageBuffer(  
    long hScanner,  
    float ratio,  
    byte[] wsqData,  
    int[] wsqDataLen );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *wsqData [out]* : Pointer to WSQ format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *wsqDataLen [out]* : pointer to WSQ format image data size

Return Values([refer to return values](#))

UFS_GetCaptureImageBufferToWSQImageBufferVar

Copies the capture image buffer (cropped or expanded to the specified size) to the target image data buffer of the WSQ format.

```
int UFS_GetCaptureImageBufferToWSQImageBufferVar(  
    long hScanner,  
    float ratio,  
    byte[] wsqData,  
    int[] wsqDataLen,  
    int nWidth,  
    int nHeight );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *wsqData [out]* : Pointer to WSQ format image data; The buffer must be allocated bigger than the size of capture image buffer in advance
- *wsqDataLen [out]* : pointer to WSQ format image data size
- *nWidth [in]* : Width to resize the capture image
- *nHeight [in]* : Height to resize the capture image

Return Values([refer to return values](#))

UFS_DecompressWSQBMP

Decompress WSQ file and save to BMP file.

```
int UFS_ DecompressWSQBMP(  
    long hScanner,  
    String wsqFile,  
    String bmpFile );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *wsqFile [in]* : Specifies file name to get wsq data buffer
- *bmpFile [in]* : Specifies file name to save image buffer

Return Values([refer to return values](#))

UFS_DecompressWSQBMPMem

Decompress WSQ buffer and save to image data of bmp format.

```
int UFS_ DecompressWSQBMPMem(  
    long hScanner,  
    byte[] wsqBuffer,  
    int wsqBufferLen,  
    byte[] bmpBuffer,  
    int[] bmpBufferLen );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *wsqBuffer [in]* : Pointer to WSQ format image data
- *wsqBufferLen [in]* : Size of WSQ format image data
- *bmpBuffer [out]* : Pointer to bmp image data; The array must be allocated bigger than the size of capture image buffer in advance.
- *bmpBufferLen [out]* : pointer to bmp image data size

Return Values([refer to return values](#))

UFS_SaveCaptureImageBufferToBMP

Saves the capture image buffer to the specified file of the bitmap format.

```
int UFS_SaveCaptureImageBufferToBMP(  
    long hScanner,  
    String szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szFileName [in]* : Specifies file name to save image buffer

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
String szFileName;  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Get file name, szFileName  
  
ufs_res = p.UFS_SaveCaptureImageBufferToBMP(hScanner[], szFileName);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferToBMP is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferToBMP is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values^(refer to return values)

UFS_SaveCaptureImageBufferTo19794_4

Saves the capture image buffer to the specified file of the 19794_4 format.

```
java int UFS_SaveCaptureImageBufferTo19794_4(  
    long hScanner,  
    String szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szFileName [in]* : Specifies file name to save image buffer

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
String szFileName;  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
// Get file name, szFileName  
  
ufs_res = p.UFS_SaveCaptureImageBufferTo19794_4(hScanner[], szFileName);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_SaveCaptureImageBufferTo19794_4 is succeeded  
}  
else  
{  
    // UFS_SaveCaptureImageBufferTo19794_4 is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_SaveCaptureImageBufferToWSQ

Saves the capture image buffer to the specified file of the WSQ format.

```
int UFS_SaveCaptureImageBufferToWSQ(  
    long hScanner,  
    float ratio,  
    String szFileName );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *szFileName [in]* : Specifies file name to save image buffer

Return Values([refer to return values](#))

UFS_SaveCaptureImageBufferToWSQVar

Saves the capture image buffer (cropped or expanded to the specified size) to the target file of the WSQ format.

```
int UFS_SaveCaptureImageBufferToWSQVar(  
    long hScanner,  
    float ratio,  
    String szFileName,  
    int nWidth,  
    int nHeight );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ratio [in]* : Compression ratio of image
- *szFileName [in]* : Specifies file name to save image buffer
- *nWidth [in]* : Width to resize the capture image
- *nHeight [in]* : Height to resize the capture image

Return Values([refer to return values](#))

UFS_ClearCaptureImageBuffer

Clears the capture image buffer stored to the internal buffer.

```
int UFS_ClearCaptureImageBuffer(  
    long hScanner);
```

Parameters

- *hScanner [in]* : Handle to the scanner object

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
ufs_res = p.UFS_ClearCaptureImageBuffer(hScanner[0]);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_ClearCaptureImageBuffer is succeeded  
}  
else  
{  
    // UFS_ClearCaptureImageBuffer is failed  
    // Use UFS_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetErrorString

Gets the error string for specified [UFS_STAUS](#) value.

```
int UFS_GetErrorString(  
    int res,  
    byte[] szErrorString );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *szErrorString [out]* : Receives error string

Examples

```
int ufs_res;  
byte[] strError = new byte[128];  
  
// Get status return code, ufs_res  
  
ufs_res = p.UFS_GetErrorString(ufs_res, strError);  
if (ufs_res == p.UFS_OK)  
{  
    // UFS_GetErrorString is succeeded  
}  
else  
{  
    // UFS_GetErrorString is failed  
}
```

Return Values[\(refer to return values\)](#)

UFS_GetTemplateType

Gets the template type value.

```
int UFS_GetTemplateType(  
    long hScanner,  
    int[] nTemplateType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTemplateType [out]* : Receives the parameter value of specified parameter type; 'pValue' must point to adequate type that is matched with the parameter type

Template type	Code	Description
UFS_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFS_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFS_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Return Values([refer to return values](#))

UFS_SetTemplateType

Sets the template type value.

```
int UFS_SetTemplateType(  
    long hScanner,  
    int nTemplateType );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nTemplateType [in]* : Parameter type; one of template type

Template type	Code	Description
UFS_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFS_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFS_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
int ufs_res;  
long[] hScanner = new long[1];  
int nTemplateType;  
  
// make class library instance(BioMiniSDK p)  
  
// Get hScanner handle  
  
nTemplateType =UFS_TEMPLATE_TYPE_SUPREMA;  
  
ufs_res = p.UFS_SetTemplateType(hScanner[], nTemplateType);  
// Error handling routine is omitted
```

Return Values(refer to return values)

UFS_SelectTemplate

Selects n number of good templates from m number of input templates.

```
int UFS_SelectTemplate(  
    long hScanner,  
    byte[][] ppTemplateInput,  
    int[] pnTemplateInputSize,  
    int nTemplateInputNum,  
    byte[][] ppTemplateOutput,  
    int[] pnTemplateOutputSize,  
    int nTemplateOutputNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *ppTemplateInput [in]* : Array pointer to the input template arrays
- *pnTemplateInputSize [in]* : Array pointer to input templates'size
- *nTemplateInputNum [in]* : Number of input templates
- *ppTemplateOutput [out]* : Array pointer to the output template arrays
- *pnTemplateOutputSize [out]* : Array pointer to the output templates'size
- *nTemplateOutputNum [in]* : Number of output templates; should be less than input template number by more than one

Return Values([refer to return values](#))

UFS_SelectTemplateEx

Selects n number of good templates from m number of input templates. This is extended version of [UFS_SelectTemplate](#) function to accommodate large size template.

```
nt UFS_SelectTemplateEx(  
    long hScanner,  
    int nBufferSize,  
    byte[][] ppTemplateInput,  
    int[] pnTemplateInputSize,  
    int nTemplateInputNum,  
    byte[][] ppTemplateOutput,  
    int[] pnTemplateOutputSize,  
    int nTemplateOutputNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *nBufferSize [in]* : Template buffer size
- *ppTemplateInput [in]* : Array pointer to the input template arrays
- *pnTemplateInputSize [in]* : Array pointer to the input templates'size
- *nTemplateInputNum [in]* : Number of input templates
- *ppTemplateOutput [out]* : Array pointer to the output template arrays
- *pnTemplateOutputSize [out]* : Array pointer to the output templates'size
- *nTemplateOutputNum [in]* : Number of output templates; should be less than input template number by more than one

Return Values([refer to return values](#))

UFS_GetFPQuality

Calculates the quality score of an image as defined in NISTIR 7151: FingerPrint Image Quality. The score would be between 1(excellent) and 5(poor).

```
int UFS_GetFPQuality(  
    long hScanner,  
    byte[] pFPImage,  
    int nWidth,  
    int nHeight,  
    int[] pnFPQuality );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pFPImage [in]* : Raw capture image data
- *nWidth [in]* : Width of capture image data
- *nHeight [in]* : Height of capture image data
- *pnFPQuality [in]* : NIST quality score of image data

Return Values([refer to return values](#))

UFS_GetFeatureNumber

Get the number of Minutiae from the template data.

```
int UFS_GetFeatureNumber(  
    long hScanner,  
    byte[] pTemplate,  
    int nTemplateSize,  
    int[] pnFeatureNum );
```

Parameters

- *hScanner [in]* : Handle to the scanner object
- *pTemplate [in]* : Template data
- *nTemplateSize [in]* : Size of template data
- *pnFeatureNum [out]* : The number of minutiae from pTemplate

Return Values([refer to return values](#))

UFM_Create

Creates a matcher object.

```
int UFM_Create(  
    long hMatcher);
```

Parameters

- *hMatcher [out]* : Pointer to handle of the matcher object

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
  
//make instance  
p = new BioMiniSDK();  
  
ufm_res = p.UFM_Create(hMatcher);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_Create is succeeded  
    // hMatcher[0] is the handle  
}  
else  
{  
    // UFM_Create is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values([refer to return values](#))

UFM_Delete

Deletes specified matcher object.

```
int UFM_Delete(  
    long hMatcher);
```

Parameters

- *hMatcher [in]* : Handle to the matcher object

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
ufm_res = p.UFM_Delete(hMatcher[0]);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_Delete is succeeded  
}  
else  
{  
    // UFM_Delete is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFM_GetParameter

Gets parameter value of UFMatcher module.

```
int UFM_GetParameter(  
    long hMatcher,  
    int nParam,  
    int[] pValue );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFM_PARAM_FAST_MODE	301	Fast Mode (0: not use fast mode, 1: use fast mode)	1
UFM_PARAM_SECURITY_LEVEL	302	Set the False Accept Ratio(FAR) (1: Below 1%, 2: Below 0.1%, 3: Below 0.01%, 4: Below 0.001%, 5: Below 0.0001%, 6: Below 0.00001%, 7: Below 0.000001%)	4
UFM_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFM_PARAM_AUTO_ROTATE	321	Rotate Mode(0: not use rotate mode, 1: use rotate mode)	0
UFM_PARAM_SDK_VERSION	210	SDK Version (get only)	-
UFM_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	

- *pValue [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
int[] nValue = new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Get fast mode  
ufm_res = p.UFM_GetParameter(hMatcher[], p.UFM_PARAM_FAST_MODE, nValue);  
// Error handling routine is omitted  
  
// Get security level  
ufm_res = p.UFM_GetParameter(hMatcher[], p.UFM_PARAM_SECURITY_LEVEL,  
nValue);  
// Error handling routine is omitted  
  
// Get use SIF  
ufm_res = p.UFM_GetParameter(hMatcher[], p.UFM_PARAM_USE_SIF, &nValue);  
// Error handling routine is omitted
```

Return Values([refer to return values](#))

UFM_SetParameter

Sets parameter value of UFMatcher module.

```
int UFM_SetParameter(  
    long hMatcher,  
    int nParam,  
    int[] pValue );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *nParam [in]* : Parameter type; one of parameters

Parameter	Code	Description	Default value
UFM_PARAM_FAST_MODE	301	Fast Mode (0: not use fast mode, 1: use fast mode)	1
UFM_PARAM_SECURITY_LEVEL	302	Set the False Accept Ratio(FAR) (1: Below 1%, 2: Below 0.1%, 3: Below 0.01%, 4: Below 0.001%, 5: Below 0.0001%, 6: Below 0.00001%, 7: Below 0.000001%)	4
UFM_PARAM_USE_SIF	311	Use SIF (0: not use SIF, 1: use SIF)	0
UFM_PARAM_AUTO_ROTATE	321	Rotate Mode(0: not use rotate mode, 1: use rotate mode)	0
UFM_PARAM_SDK_VERSION	210	SDK Version (get only)	-
UFM_PARAM_SDK_COPYRIGHT	211	SDK Copyright (get only)	

- *pValue [in]* : Pointer to parameter value of specified parameter type; pValue must point to adequate storage type matched to parameter type

Examples

```
int_STATUS ufm_res;  
long[] hMatcher = new long[1];  
int[] nValue = new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Set fast mode to nValue  
nValue[] = 1;  
ufm_res = p.UFM_SetParameter(hMatcher[], p.UFM_PARAM_FAST_MODE, nValue);  
// Error handling routine is omitted  
  
// Set security level to nValue  
nValue[] = 4;  
ufm_res = UFM_SetParameter(hMatcher[], p.UFM_PARAM_SECURITY_LEVEL, nValue);  
// Error handling routine is omitted  
  
// Set use SIF to nValue  
nValue[] = ;
```

```
ufm_res = UFM_SetParameter(hMatcher[], p.UFM_PARAM_USE_SIF, nValue);  
// Error handling routine is omitted
```

Return Values([refer to return values](#))

UFM_Verify

Compares two extracted templates.

```
int UFM_Verify(  
    long hMatcher,  
    byte[] pTemplate1,  
    int nTemplate1Size,  
    byte[] pTemplate2,  
    int nTemplate2Size,  
    int[] bVerifySucceed );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pTemplate1 [in]* : Pointer to the template1
- *nTemplate1Size [in]* : Specifies the size of the template1
- *pTemplate2 [in]* : Pointer to the template2
- *nTemplate2Size [in]* : Specifies the size of the template2
- *bVerifySucceed [out]* : Receives, whether verification is succeed; 1: verification is succeed, 0: verification is failed

Examples

```
// Assume template size is 1024 bytes  
int MAX_TEMPLATE_SIZE = 1024;  
  
int ufm_res;  
Pointer hMatcher;  
byte[] Template1 = new byte[MAX_TEMPLATE_SIZE];  
byte[] Template2 = new byte[MAX_TEMPLATE_SIZE];  
int nTemplate1Size;  
int nTemplate2Size;  
int[] bVerifySucceed = new int[1];  
int nSucceed;  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Get two templates, Template1 and Template2  
  
ufm_res = p.UFM_Verify(hMatcher[], Template1, nTemplate1Size, Template2,  
nTemplate2Size, bVerifySucceed);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_Verify is succeeded  
    nSucceed=bVerifySucceed[];
```

```
if (nSucceed)
{
    // Template1 is matched to Template2
}
else
{
    // Template1 is not matched to Template2
}
}
else
{
    // UFM_Verify is failed
    // Use UFM_GetErrorString function to show error string
}
```

Return Values([refer to return values](#))

UFM_Identify, UFM_IdentifyMT

Compares a template with given template array. UFM_IdentifyMT function uses multi threads internally for faster identifying in multi-core systems.

```
int UFM_Identify(  
    long hMatcher,  
    byte[] pTemplate1,  
    int nTemplate1Size,  
    byte[][] pTemplate2,  
    int[] nTemplate2Size,  
    int nTemplate2Num,  
    int nTimeout,  
    int[] nMatchTemplate2Index );  
  
int UFM_IdentifyMT(  
    long hMatcher,  
    byte[] pTemplate1,  
    int nTemplate1Size,  
    byte[][] pTemplate2,  
    int[] nTemplate2Size,  
    int nTemplate2Num,  
    int nTimeout,  
    int[] nMatchTemplate2Index );
```

Parameters

- *hMatcher [in]* : Handle of the matcher object
- *pTemplate1 [in]* : Pointer to the template
- *nTemplate1Size [in]* : Specifies the size of the template
- *ppTemplate2 [in]* : Pointer to the template array
- *pnTemplate2Size [in]* : Pointer to the template size array
- *nTemplate2Num [in]* : Specifies the number of templates in the template array
- *nTimeout [in]* : Specifies maximum time for identifying in milliseconds; If elapsed time for identifying exceeds nTimeout, function stops further identifying and returns UFM_ERR_MATCH_TIMEOUT; 0 means infinity
- *pnMatchTemplate2Index [out]* : Receives the index of matched template in the template array; -1 means pTemplate1 is not matched to all of templates in ppTemplate2

Return Values(refer to [return values](#))

UFM_AbortIdentify

Aborts current identifying procedure started using [UFM_Identify](#).

```
int UFM_AbortIdentify(  
    long hMatcher);
```

Parameters

- *hMatcher [in]* : Handle to the matcher object

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Start UFM_Identify  
  
ufm_res = p.UFM_AbortIdentify(hMatcher);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_AbortIdentify is succeeded  
}  
else  
{  
    // UFM_AbortIdentify is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFM_IdentifyInit

Initializes identify with input template.

```
int UFM_IdentifyInit(  
    long hMatcher,  
    byte[] pTemplate1,  
    int nTemplate1Size );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pTemplate1 [in]* : Pointer to the template
- *nTemplate1Size [in]* : Specifies the size of the template

Examples

```
// Assume template size is 1024 bytes  
int MAX_TEMPLATE_SIZE = 1024;  
  
int ufm_res;  
long[] hMatcher = new long[1];  
byte[] Template1 = new byte[MAX_TEMPLATE_SIZE];  
int nTemplate1Size;  
  
// make class library instance(BioMiniSDK p)  
  
    // Create hMatcher handle  
  
// Get Template1  
  
ufm_res = p.UFM_IdentifyInit(hMatcher[], Template1, nTemplate1Size);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_IdentifyInit is succeeded  
}  
else  
{  
    // UFM_IdentifyInit is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values(refer to return values)

UFM_IdentifyNext

Matches one input template to the template specified in [UFM_IdentifyInit](#).

```
int UFM_IdentifyNext(  
    long hMatcher,  
    byte[] pTemplate2,  
    int nTemplate2Size,  
    int[] bIdentifySucceed );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pTemplate2 [in]* : Pointer to the template array
- *nTemplate2Size [in]* : Specifies the size of the template array
- *bIdentifySucceed [out]* : Receives whether identification is succeed; 1: identification is succeed, 0: identification is failed

Examples

```
int MAX_TEMPLATE_NUM = 50;  
  
int ufm_res;  
long[] hMatcher = new long[1];  
byte[] Template2 = new byte[MAX_TEMPLATE_NUM];  
int[] nTemplate2Size = new int[MAX_TEMPLATE_NUM];  
int nTemplate2Num;  
int[] bIdentifySucceed = new int[1];  
int i;  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Get number of templates in DB or something, and save it to nTemplate2Num  
  
for (i = 0; i < nTemplate2Num; i++)  
{  
    // Get one template in DB or something, and save it to Template2 and  
    // nTemplate2Size  
  
    ufm_res = p.UFM_IdentifyNext(hMatcher[i], Template2[i], nTemplate2Size[i],  
    bIdentifySucceed[i]);  
    if (ufm_res == p.UFM_OK)  
    {  
        // UFM_IdentifyNext is succeeded  
    }  
    else
```

```

{
    // UFM_IdentifyNext is failed
    // Use UFM_GetErrorString function to show error string
    // return;
}

if (bIdentifySucceed[])
{
    // Identification is succeed
    break;
}
}
if (!bIdentifySucceed[])
{
    // Identification is failed
}

```

Return Values(refer to return values)

UFM_RotateTemplate

Rotates the specified template to the amount of 180 degrees.

```
int UFM_RotateTemplate(  
    long hMatcher,  
    byte[] pTemplate,  
    int nTemplateSize );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pTemplate [in / out]* : Pointer to the template
- *nTemplateSize [in]* : Specifies the size of the template

Examples

```
// Assume template size is 1024 bytes  
int MAX_TEMPLATE_SIZE = 1024;  
  
int ufm_res;  
long[] hMatcher = new long[1];  
byte[] Template = new byte[MAX_TEMPLATE_SIZE];  
int nTemplateSize;  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
// Get a template, and save it to Template and nTemplateSize  
  
ufm_res = p.UFM_RotateTemplate(hMatcher[], Template, nTemplateSize);  
if (ufm_res == p.UFM_OK)  
{  
    // UFM_RotateTemplate is succeeded  
}  
else  
{  
    // UFM_RotateTemplate is failed  
    // Use UFM_GetErrorString function to show error string  
}
```

Return Values[\(refer to return values\)](#)

UFM_GetErrorString

Gets the error string for specified [UFM_STATUS](#) value.

```
int UFM_GetErrorString(  
    int res,  
    byte[] szErrorString );
```

Parameters

- *res [in]* : Status return value
- *szErrorString [out]* : Receives error string

Examples

```
int ufs_res;  
byte[] strError = new byte[128];  
  
// Get status return code, ufm_res  
ufs_res = p.UFM_GetErrorString(ufs_res, strError);  
if (ufs_res == p.UFS_OK)  
{  
    // UFM_GetErrorString is succeeded  
}  
else  
{  
    // UFM_GetErrorString is failed  
}
```

Return Values[\(refer to return values\)](#)

UFM_GetTemplateType

Gets the parameter value.

```
int UFM_GetTemplateType(  
    long hMatcher,  
    int[] nTemplateType );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *pValue [out]* : Receives parameter value of specified parameter type; pValue must point to adequate storage type matched to template type

Template type	Code	Description
UFM_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFM_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFM_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
int[] nTemplateType = new int[1];  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
  
ufm_res = p.UFM_GetTemplateType(hMatcher[], nTemplateType );  
// Error handling routine is omitted
```

Return Values[\(refer to return values\)](#)

UFM_SetTemplateType

Gets parameter value.

```
int UFM_SetTemplateType(  
    long hMatcher,  
    int nTemplateType );
```

Parameters

- *hMatcher [in]* : Handle to the matcher object
- *nTemplateType [in]* : Parameter type; one of template type

Template type	Code	Description
UFM_TEMPLATE_TYPE_SUPREMA	2001	Suprema template type
UFM_TEMPLATE_TYPE_ISO19794_2	2002	ISO template type
UFM_TEMPLATE_TYPE_ANSI378	2003	ANSI378 template type

Examples

```
int ufm_res;  
long[] hMatcher = new long[1];  
int nTemplateType;  
  
// make class library instance(BioMiniSDK p)  
  
// Create hMatcher handle  
nTemplateType =p.UFM_TEMPLATE_TYPE_SUPREMA;  
  
ufm_res = p.UFM_SetTemplateType(hMatcher[], nTemplateType);  
// Error handling routine is omitted
```

Return Values[\(refer to return values\)](#)

UFS_STATUS

Every function in a UFSscanner module returns **UFS_OK** when it succeeds. When it fails, it returns a value corresponding to a error code. Please find the error code on the followings if you'd like to know the information about the UFS_STATUS (integer) value.

Code	Value	Description
UFS_OK	0	Success
UFS_ERROR	-1	General error
UFS_ERR_NO_LICENSE	-101	Device is not connected or License is not located
UFS_ERR_LICENSE_NOT_MATCH	-102	License does not match
UFS_ERR_LICENSE_EXPIRED	-103	License has expired
UFS_ERR_NOT_SUPPORTED	-111	This function is not supported
UFS_ERR_INVALID_PARAMETERS	-112	Input parameters are invalid
UFS_ERR_ALREADY_INITIALIZED	-201	Module is already initialized
UFS_ERR_NOT_INITIALIZED	-202	Module is not initialized
UFS_ERR_DEVICE_NUMBER_EXCEED	-203	Device number exceeds
UFS_ERR_LOAD_SCANNER_LIBRARY	-204	Error on loading the library of a scanner
UFS_ERR_CAPTURE_RUNNING	-211	Capturing is started using UFS_CaptureSingleImage or UFS_StartCapturing
UFS_ERR_CAPTURE_FAILED	-212	Capturing is timeout or aborted
UFS_ERR_NOT_GOOD_IMAGE	-301	Input image is not good
UFS_ERR_EXTRACTION_FAILED	-302	Extraction is failed
UFS_ERR_CORE_NOT_DETECTED	-351	Core is not detected
UFS_ERR_CORE_TO_LEFT_TOP	-353	Move finger to left-top
UFS_ERR_CORE_TO_TOP	-354	Move finger to top
UFS_ERR_CORE_TO_RIGHT_TOP	-355	Move finger to right-top
UFS_ERR_CORE_TO_RIGHT	-356	Move finger to right
UFS_ERR_CORE_TO_RIGHT_BOTTOM	-357	Move finger to right-bottom
UFS_ERR_CORE_TO_BOTTOM	-358	Move finger to bottom
UFS_ERR_CORE_TO_LEFT_BOTTOM	-359	Move finger to left-bottom

UFM_STATUS

Every function in a UFMatcher module returns **UFM_OK** when it succeeds. When it fails, it returns a value corresponding to a error code. Please find the error code on the followings if you'd like to know the information about the UFM_STATUS (integer) value.

Code	Value	Description
UFM_OK	0	Success
UFM_ERROR	-1	General error
UFM_ERR_NO_LICENSE	-101	System has no license
UFM_ERR_LICENSE_NOT_MATCH	-102	License does not match
UFM_ERR_LICENSE_EXPIRED	-103	License has expired
UFM_ERR_NOT_SUPPORTED	-111	This function is not supported
UFM_ERR_INVALID_PARAMETERS	-112	Input parameters are invalid
UFM_ERR_MATCH_TIMEOUT	-401	Matching is timeout
UFM_ERR_MATCH_ABORTED	-402	Matching is aborted
UFM_ERR_TEMPLATE_TYPE	-411	Template type does not match

5. Appendix

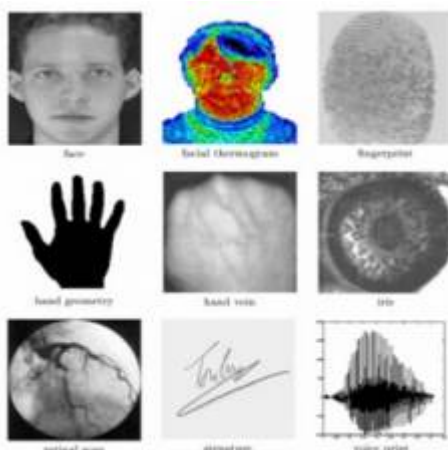
What is Biometrics?



Identifying individuals based on their distinctive anatomical(fingerprint, face, iris, hand geometry) and behavioral(signature, voice) characteristics is called biometrics. Because biometric identifiers cannot be shared or misplaced, they intrinsically represent an individual's identify. Biometrics is quickly becoming an essential component of effective identification solutions. Recognition of a person by their body, then linking that body to an externally established "identity", forms a powerful authentication tool.

Fingerprint identification has been widely used for a long time because individual's unique Fingerprint pattern cannot be shared or misplaced, they essentially represent an individual's identity. Fingerprint contains rich information in a small area compared to other biological anatomical (face, iris, hand geometry) and behavioral (signature, voice) characteristics. Identifying individuals based on fingerprint is well-known as the most feasible method compared to other biometrical characteristics. Among the other biometric identification methods, fingerprint identification has a good balance of qualities including accuracy, throughput, size and cost of scan devices, maturity of technology and convenience of use, making it the dominant biometric technology in industry.

Two major methods are being used in fingerprint identification; matching based on feature point and filter bank. Matching algorithm based on feature point extract "local minutiae from Thinning fingerprint image or gray scale fingerprint image. Thereafter a method for matching using the location relationship between the feature point in the fingerprint template stored image and the input fingerprint image. he feature point matching takes relatively short processing time however matching performance could be decreased on noised fingerprint images. Filters bank matching algorithm extract fingerprint ridge using Gabor filters bank.



Measured by means of an automated device for physical and behavioral characteristics, Technology to utilize as a means of personal identification

Physical characteristics available - fingerprint, palm print, face, iris veins, etc. Physical and behavioral features available - voice, signature, key tapping, Gait, etc.

Fingerprint is specified patterned from finger's sweat gland which is consisting of ridge and Valley. Ridge has a certain direction at even interval. Also Ridge (Ridge) has characteristics where lean towards end from certain direction and the ridge (Ridge) part

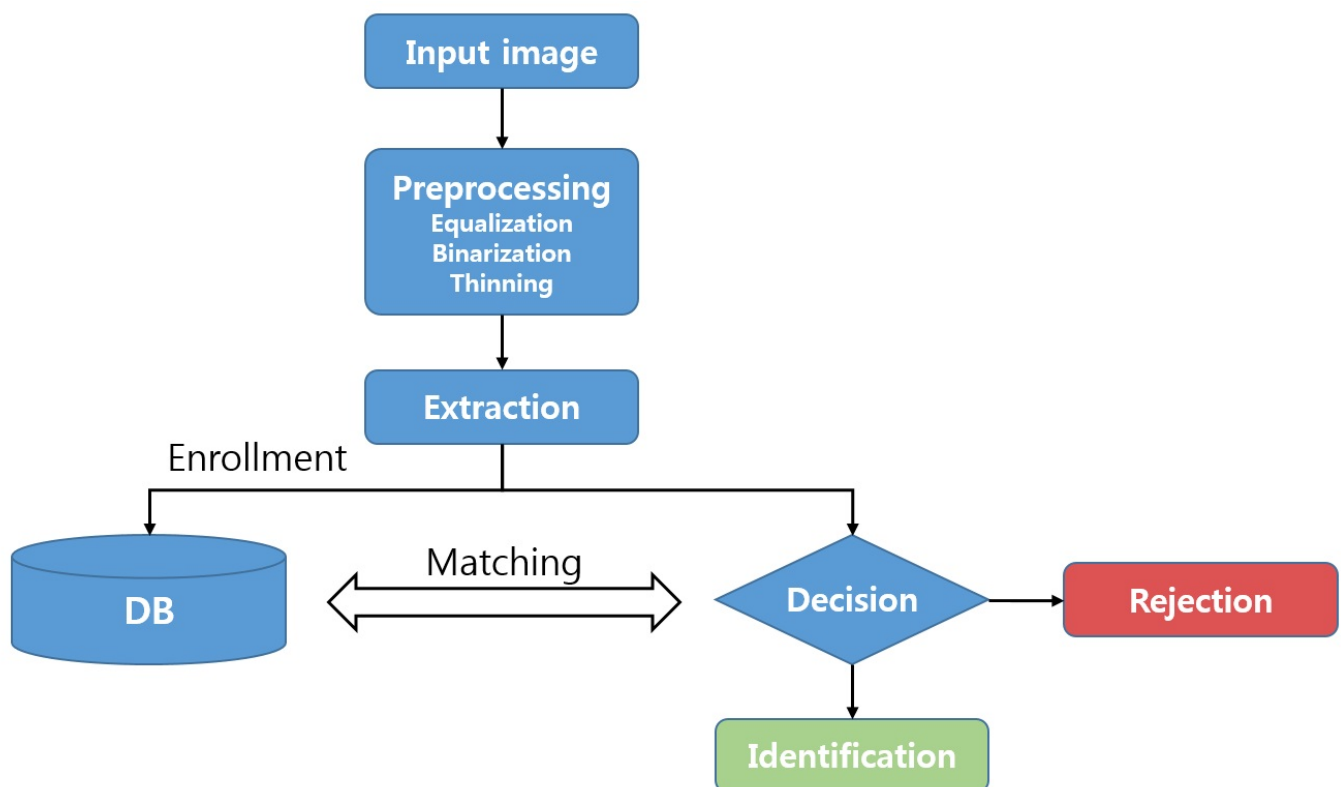
(Junction birurbation) which is divided into two directions.

Classification of fingerprint recognition

Fingerprint recognition is divided into classification (Classification) and matching (Matching). Classification (Classification) is a way of determine that fingerprint belongs to a particular group based on the overall form of a fingerprint. Matching (Matching) is to compare the fingerprints match with the fingerprints stored in the input. Matching (Matching) is divided into 2 different methods; 1: N matching (Identification) and 1: 1 matching (Verification). The extracted fingerprint minute is used to do matching. The minutiae can be divided into Local feature and global feature by its own characteristic The Global feature can be determined by formation of fingerprint and direction pattern of ridge & location of ridge minutiae determines the Local feature.

Fingerprint recognition

Basic formation of fingerprint recognition system is as below. Preprocess the fingerprint image which was entered by fingerprint reader. Then recognize the fingerprint by extracting the features and comparing with saved fingerprint data stored in database.



a. Preprocessing

To extract features from the fingerprint image, follow through below process. Clarifying the image by removing the noise and also ridge ending, minutiae placement, crossed features caused by the noise. This process is called the preprocessing. Here, 3 preprocessing methods are performed; smoothing, binarization and thinning.

b. Feature Extraction

Fingerprint features can be performed as below methods; End point of ridge, minutiae placement, ridge direction pattern, connecting information between core and ridge, local feature containing ridge formation.

The feature extraction can be effected by fingerprint pressure, direction, location, placement, and condition of the fingerprint.

c. Feature Matching

The matching is comparing saved template pattern in the database with fingerprint features taken from the device. To determine the match, may use pattern matching, statistics identification, structure identification.

About FAR, FRR and EER

Here, we discusses some general principles of biometric recognition systems, describes different classification errors and explains how the quality of two systems can be compared objectively.

a. Identification vs. Verification

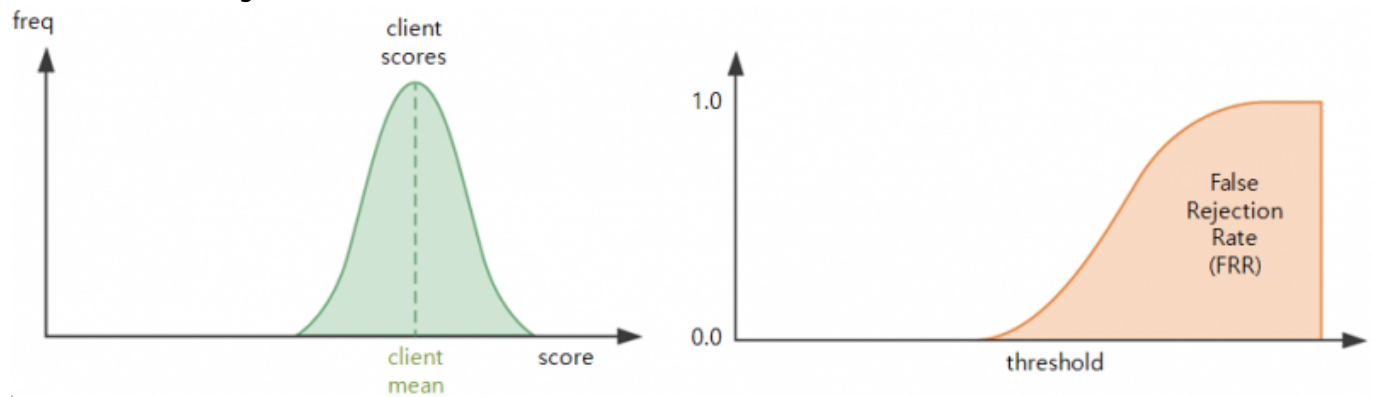
A biometric recognition system can run in two different modes: identification or verification. Identification is the process of trying to find out a person's identity by examining a biometric pattern calculated from the person's biometric features. In the identification case, the system is trained with the patterns of several persons. For each of the persons, a biometric template is calculated in this training stage. A pattern that is going to be identified is matched against every known template, yielding either a score or a distance describing the similarity between the pattern and the template. The system assigns the pattern to the person with the most similar biometric template. To prevent impostor patterns (in this case all patterns of persons not known by the system) from being correctly identified, the similarity has to exceed a certain level. If this level is not reached, the pattern is rejected. In the verification case, a person's identity is claimed a priori. The pattern that is verified only is compared with the person's individual template. Similar to identification, it is checked whether the similarity between pattern and template is sufficient to provide access to the secured system or area.

b. Thresholding (False Acceptance / False Rejection)

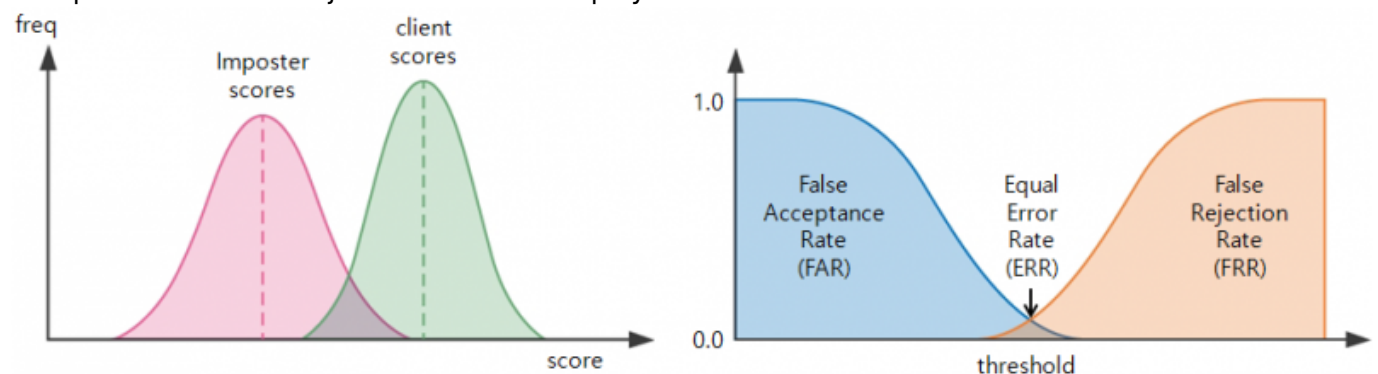
The threshold depending fraction of the falsely accepted patterns divided by the number of all impostor patterns is called **False Acceptance Rate (FAR)**. Its value is one, if all impostor patterns are falsely accepted and zero, if none of the impostor patterns is accepted. Look on the graphic on the right to see the values of the FAR for the score distribution of the left image for varying threshold.



Now let's change to the client patterns. Similar to the impostor scores, the client pattern's scores vary around a certain mean value. The mean score of the client patterns is higher than the mean value of the impostor patterns, as shown in the left of the following two images. If a classification threshold that is too high is applied to the classification scores, some of the client patterns are falsely rejected. Depending on the value of the threshold, between none and all of the client patterns will be falsely rejected. The fraction of the number of rejected client patterns divided by the total number of client patterns is called **False Rejection Rate (FRR)**. According to the FAR, its value lies in between zero and one. The image on the right shows the FAR for a varying threshold for the score distribution shown in the image on the left.



The choice of the threshold value becomes a problem if the distributions of the client and the impostor scores overlap, as shown in the next image on the left. On the right, the corresponding false acceptance and false rejection rates are displayed.



Note that if the score distributions overlap, the FAR and FRR intersect at a certain point. The value of the FAR and the FRR at this point, which is of course the same for both of them, is called the **Equal Error Rate (EER)**.

Scanners

BioMini Plus2



FBI PIV Approved Authentication Scanner

BioMini Plus 2 is an USB fingerprint scanner designed to provide high level security solution for identity access management solutions. BioMini Plus 2 features Suprema's latest live finger detection technology for protection against fake fingerprints. The device features sleek and ergonomic design with scratch resistant IP65 sensor surface securing high usability. The usability is further extended with integration of MDR technology that allows effective capture of fingerprints even under direct sunlight. Combined with its comprehensive SDK solution, BioMini Plus 2 is a versatile and optimal platform for system integrators, hardware manufacturers and security companies.

Specification

Main

Sensor Type(Optical)
Resolution(500dpi/256gray)
Sensing Area(16.0x18.0mm)
Image Size(315×354 pixels)
Compression Standards(WSQ)
Template Format(Suprema,ISO19794-2, ANSI-378)
Image Format(ISO19794-4)
IP Rating(IP65)

Interface

USB(2.0 High-speed)

Hardware

Operating Temperature(-10°C ~ 50°C)
Certification(CE,FCC,UL,WHQL,USB-IF RoHS,WEEE)
Dimensions(66x90x58mm/WxLxH)

Compatibility

Operating System(Windows, Linux, Android4.1/Jelly Bean and Above)

BioMini Slim



FBI PIV and Mobile ID FAP20 Certified USB Fingerprint Scanner

BioMini Slim has been designed to provide a high level security solution for identity access management solutions for authentication. With IP65 grade dust and waterproof form factor, BioMini Slim features a sleek ergonomic design with the latest 500dpi slim optical sensor, which boasts a large platen size for easy and reliable fingerprints capturing as well as advanced LFD (Live Finger Detection) technology.

Specification

Main

Sensor Type(Optical)
Resolution(500dpi/256gray)
Platen Size(18×25.4mm)
Sensing Area(17×25mm)
Image Size(320×480 pixels)
Compression Standards(WSQ)
Template Format(Suprema,ISO19794-2, ANSI-378)
Image Format(ISO19797-4)
IP Rating(IP65)

Interface

USB(2.0 High-speed)

Hardware

Operating Temperature(-10°C ~ 50°C)
Certification(CE,FCC,KC,UL,WHQL, USB-IF,WEEE)
Dimensions(82×57.7×27mm/WxLxH)

Compatibility

Operating System(Windows, Linux, Android4.1/Jelly Bean and Above)

BioMini Combo



Contact Smart Card Reader with USB Fingerprint Scanner

Suprema BioMini Combo has been designed to provide two factor authentication security solutions for authentication purposes. The scanner features Suprema's latest slim optical sensor with large platen size for easier capturing. Smart card reader functionality and advanced Live Finger Detection technology enhances security makes BioMini Combo a secure platform for developers.

Specification

Main

Sensor Type(Optical)
Resolution(500dpi/256gray)
Platen Size(18×25.4mm)
Sensing Area(17×25mm)
Image Size(320×480 pixels)
Compression Standards(WSQ)
Template Format(Suprema,ISO19794-2, ANSI-378)
Image Format(ISO19797-4)

Card Support

CardType(ISO 7816/EMV 2000with SAM Slot_optical)

Interface

USB(2.0 High-speed)

Hardware

Operating Temperature(-10°C ~ 50°C)
Certification(CE,FCC,KC,UL,WHQL, USB-IF,WEEE)
Dimensions(82×57.7×27mm/WxLxH)

Compatibility

Operating System(Windows, Linux)

BioMini Plus



FBI PIV Approved Authentication Scanner

Suprema BioMiniPlus has been designed to provide high level security solution with its proven reliability through FBI-PIV certification. BioMini Plus features hybrid live finger detection (LFD) technology and multi award-winning Suprema Algorithm. Packed in a sleek and ergonomic design, it features durable 500dpi optical sensor and high speed USB 2.0 interface. Combined with its comprehensive SDK solution,

Specification

Main

Sensor Type(Optical)
Resolution(500dpi/256gray)
Sensing Area(15.5×18.8mm)
Image Size(260×340 pixels)
Compression Standards(WSQ)
Template Format(Suprema,ISO19794-2, ANSI-378)
Image Format(ISO19797-4)
IP Rating(IP65)

Interface

USB(2.0 High-speed)

Hardware

Operating Temperature(-10°C ~ 50°C)
Certification(CE,FCC,KC,UL,WHQL, USB-IF,WEEE)
Dimensions(66x90x58mm/WxLxH)

Compatibility

Operating System(Windows, Linux, Android4.1/Jelly Bean and Above)

BioMini



High Performance Authentication Scanner Our latest range of high performance fingerprint scanners are supported by powerful software development kit (SDK) which features Suprema's award-winning algorithm that ranked 1st in FVC and NIST MINEX tests. Suprema Authentication Scanners offers unrivaled versatile platform for development to leading security companies, system integrators and hardware manufacturers.

Specification

Main

Sensor Type(Optical)
Resolution(500dpi/256gray)
Sensing Area(16x18mm)
Image Size(288×320 pixels)
Compression Standards(WSQ)
Template Format(Suprema,ISO19794-2, ANSI-378)
Image Format(ISO19797-4)

Interface

USB(2.0 High-speed)

Hardware

Operating Temperature(-10°C ~ 50°C)
Certification(CE,FCC,KC,WHQL)
Dimensions(66x90x58mm/WxLxH)

Compatibility

Operating System(Windows, Linux, Android4.1/Jelly Bean and Above)