



Programming Manual for FDx SDK Pro for Windows

For applications using SecuGen® fingerprint readers

SG1-0030A-026 (Last updated: 05/30/2024)

Copyright © 1998-2024 SecuGen Corporation. ALL RIGHTS RESERVED. Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used only in accordance with the terms of the agreement. SecuGen, Auto-On, FDU02, FDU03, FDU04, SDU03, SDU04, U10, U20, U30, UPx, U-Air and U20-ASF-BT are trademarks or registered trademarks of SecuGen Corporation. All other brands or product names may be trademarks, service marks or registered trademarks of their respective owners.

Contents

CHAPTER 1. OVERVIEW	4
1.1. FEATURES	4
1.2. SYSTEM REQUIREMENTS	4
CHAPTER 2. FILES INCLUDED WITH THE SDK	5
2.1. INCLUDED FILES	5
CHAPTER 3. PROGRAMMING IN C/C++	7
3.1. CREATING SGFPM	7
3.2. INITIALIZING SGFPM	7
3.3. TERMINATING SGFPM	8
3.4. OPENING THE SECUGEN FINGERPRINT READER	9
3.5. GETTING DEVICE INFORMATION	9
3.6. CAPTURING A FINGERPRINT IMAGE	10
3.7. GETTING IMAGE QUALITY	12
3.8. CONTROLLING BRIGHTNESS	12
3.9. CREATING A TEMPLATE	12
3.10. MATCHING TEMPLATES	13
3.11. REGISTRATION PROCESS	16
3.12. VERIFICATION PROCESS	17
3.13. GETTING MATCHING SCORE	17
3.14. USING AUTO-ON™	18
3.15. TEMPLATE FORMAT	19
3.16. MANIPULATING ANSI378 TEMPLATES	21
3.17. MANIPULATING ISO19794-2 TEMPLATES	23
3.18. MANIPULATING ISO19794-2 COMPACT CARD TEMPLATES	25
3.19. GETTING VERSION INFORMATION OF MINEX COMPLIANT ALGORITHMS	28
CHAPTER 4. FUNCTION REFERENCE	29
4.1. SGFPM CREATION AND TERMINATION	29
4.2. INITIALIZATION	29
4.3. DEVICE AND CAPTURING FUNCTIONS	31
4.4. EXTRACTION FUNCTIONS	37
4.5. MATCHING FUNCTIONS	39
4.6. FUNCTIONS FOR ANSI378 TEMPLATES	41
4.7. FUNCTIONS FOR ISO19794-2 TEMPLATES	44
4.8. FUNCTIONS FOR ISO19794-2 COMPACT CARD TEMPLATES	47
4.9. OTHER	49
CHAPTER 5. STRUCTURE REFERENCE	50
5.1. SGDEVICEINFOPARAM	50
5.2. SGDEVICEINFO	51
5.3. SGDEVICELIST	51
5.4. SGFINGERINFO	51
5.5. SGANSITEMPLATEINFO/SGISOTEMPLATEINFO	52
CHAPTER 6. CONSTANTS	53
6.1. SGFDXDEVICENAME	53
6.2. SGPPPORTADDR	53
6.3. SGFDXSECURITYLEVEL	53
6.4. SGFDXTEMPLATEFORMAT	54
6.5. SGIMPRESSIONTYPE	54
6.6. SGFINGERPOSITION	54
6.7. SGFDXERRORCODE	55
6.8. OTHER CONSTANTS	56

APPENDIX A. USING SGFPM OBJECTS DIRECTLY	57
A.1. CREATING AN SGFPM OBJECT	57
A.2. DESTROYING AN SGFPM OBJECT	57
A.3. ACCESSING OTHER MEMBER FUNCTIONS	57
APPENDIX B. USING .NET LIBRARY	59
APPENDIX C. GETTING A TEMPLATE FROM BLUETOOTH DEVICES (U20-ASF-BT)	60

Chapter 1. Overview

SecuGen's FDx SDK Pro is designed to provide low level access to SecuGen's fingerprint readers using SecuGen's next-generation algorithm module. Programming with SecuGen's FDx SDK Pro is simple and easy and gives the most development flexibility among all SecuGen SDKs.

1.1. Features

- **SecuGen's new and improved next-generation algorithms**
- **Highly flexible and easy to use for developers of all kinds of applications**
- **Support for four kinds of fingerprint minutiae formats** (also known as templates):
 - SG400: SecuGen's proprietary fingerprint minutiae format
 - ANSI378: Finger Minutiae Format for Data Exchange (ANSI-INCITS 378-2004)
 - ISO19794-2: Finger Minutiae Data (ISO/IEC 19794-2:2005)
 - ISO19794-2 Compact: Finger Minutiae Data—Compact Card Template (ISO/IEC 19794-2:2005 / Section 8.2 Compact Size Finger Minutiae Format)
- **Low-level APIs for image capture, feature extraction and matching**
 - The following extraction and matching algorithms, which are incorporated in sgfpamx.dll in this SDK, support the ANSI-INCITS 378-2004 standard and have been tested and proven to be MINEX Compliant (<http://fingerprint.nist.gov/MINEX>):
 - SecuGen ANSI INCITS 378 Template Generator v3.5 (feature extraction algorithm)
 - SecuGen ANSI INCITS 378 Template Matcher v3.5 (matching algorithm)

1.2. System Requirements

SecuGen USB Fingerprint Readers capture a fingerprint image and digitize the image to an 8-bit gray-scale image at 500 DPI resolution. The host system then retrieves the image through its USB port for subsequent processing. All SecuGen USB readers, except for FDU01-based readers, are supported in this SDK.

SecuGen Bluetooth Fingerprint Readers capture a fingerprint image and digitize the image to an 8-bit gray-scale image at 500 DPI resolution. While the host system is capable of retrieving the image wirelessly for subsequent processing, it is recommended to process the image within the Bluetooth device before transmitting to the host. Refer to [Appendix C. Getting a Template from Bluetooth Devices](#) for more information. All SecuGen Bluetooth readers are supported in this SDK.

Windows System requirements

Intel Pentium Processor or faster

At least 64 MB RAM

At least 80 MB available hard disk space

For USB fingerprint readers only:

USB port (2.0 or higher)

Windows 11, 10, 8.1, 8, 7 / Vista / XP, Windows Server 2022, 2019, 2016 / 2012 / 2008 R2

For Bluetooth fingerprint readers only:

Bluetooth compatibility

Windows 10 or higher

Chapter 2. Files Included with the SDK

2.1. Included Files

After unzipping the SDK, the following files are available to use.

Bin\i386 directory

Runtime modules for 32-bit platform (same files that are copied to Windows system directory)

sgfplib.dll	Main module
sgfpamx.dll	Fingerprint algorithm module for extraction & matching (MINEX Compliant)
sgwsqplib.dll	WSQ module
sgfdusda.dll	Module for U20-ASF-BT devices (Bluetooth SPP and BLE)
sgbledev.dll	Module for U20-ASF-BT devices (Bluetooth BLE only)

Bin\x64 directory

Runtime modules for 64-bit platform

sgfplib.dll	Main module
sgfpamx.dll	Fingerprint algorithm module for extraction & matching (MINEX Compliant)
sgwsqplib.dll	WSQ module
sgfdusdax64.dll	Module for U20-ASF-BT devices (Bluetooth SPP and BLE)
sgbledev.dll	Module for U20-ASF-BT devices (Bluetooth BLE only)

Inc directory

SDK library header file

sgfplib.h	Declarations of function prototypes and structures used in the SDK
------------------	--

Lib\i386 directory

SecuGen Fingerprint Module library for 32-bit platforms

sgfplib.lib	Fingerprint Module import library
--------------------	-----------------------------------

Lib\x64 directory

SecuGen Fingerprint Module library for 64-bit platforms

sgfplib.lib	Fingerprint Module import library
--------------------	-----------------------------------

Samples directory

Visual C++ Sample Source Code

Device Test	Sample code for image capture
Matching	Sample code for template matching SecuGen proprietary templates
MatchingANSI	Sample code for matching ANSI378 templates
MatchingISO	Sample code for matching ISO19794-2 templates
MatchingUAIR	Sample code for image capture from contactless device, U-Air

.NET and .NET Framework

DotNet\Bin\i386 and DotNet\Bin\x64 directories

SecuGen.FDxSDKPro.DotNet.Windows.dll

.NET assembly file wrapping sgfplib.dll. (Note: x64 .NET dll must be used for x64 platforms)

DotNet\Samples directory

Sample code using SecuGen.FDxSDKPro.DotNet.Windows.dll

Matching sample written in C#

Matching/capture sample for contactless device, U-Air in C#

DotNetFramework\Bin\i386 and DotNetFramework\Bin\x64 directories

SecuGen.FDxSDKPro.Windows.dll

.NET assembly file wrapping sgfplib.dll. (Note: x64 .NET dll must be used for x64 platforms)

DotNetFramework\Samples directory

Sample code using SecuGen.FDxSDKPro.Windows.dll

Matching sample written in C#

Matching sample showing ANSI378 template usage written in C#

Matching/capture sample for contactless device, U-Air in C#

Matching sample written in Visual Basic .NET

Chapter 3. Programming in C/C++

SecuGen's FDx SDK Pro was designed for ease in programming and the most flexibility for developers. All SDK functions are integrated into the **SGFPM (SecuGen FingerPrint Module)** class. The SGFPM class includes device initialization, fingerprint capture, minutiae extraction and matching functions. The developer can access SDK functions directly through the SGFPM class or through C functions that wrap the SGFPM class. C functions provide access to SDK functionalities through an SGFPM handle. In this chapter, C functions are explained. For direct access to the SGFPM class, refer to [Appendix A](#).

3.1. Creating SGFPM

To use SGFPM, call **SGFPM_Create()**, which creates an SGFPM object and returns a handle to the SGFPM object. When calling **SGFPM_Create()**, pass a pointer to the handle to contain the SGFPM handle as a parameter. The SGFPM handle is used for the duration of the session to access other functions.

```
HSGFPM    m_hFpm; // handle for SGFPM
DWORD err = SGFPM_Create(&m_hFpm);
```

3.2. Initializing SGFPM

If an SGFPM object is created, it should be initialized using **SGFPM_Init()** or **SGFPM_InitEx2()**. **SGFPM_Init()** takes the device name, loads the driver that corresponds to the device name and initializes the fingerprint algorithm module based on device information. **SGFPM_InitEx2()** takes image width, image height, resolution and path to a license file as parameters. Call **SGFPM_InitEx2()** when using the fingerprint algorithm module without a SecuGen reader.

The table below summarizes the correlation among device name (device type), loaded device driver and initial image size when the **Init(SGFPMDeviceName devName)** function is called.

Device Name, Device Driver and Image Size

Device Name	Value	Device driver	Image Size (pixels)
SG_DEV_FDU02	3	FDU02 USB driver	260*300
SG_DEV_FDU03	4	FDU03 / SDU03 USB driver	260*300
SG_DEV_FDU04	5	FDU04 / SDU04 USB driver	258*336
SG_DEV_FDU05	6	U20 USB driver	300*400
SG_DEV_FDU06	7	UPx USB driver	260*300
SG_DEV_FDU06AP	22	UPx-AP USB driver	300*400
SG_DEV_FDU07	8	U10 USB driver	252*330
SG_DEV_FDU08	10	U20-A USB driver	300*400
SG_DEV_FDU08A	17	U20-AP USB driver	300*400
SG_DEV_FDU08AL	23	U20-AL USB driver	300*400
SG_DEV_FDU09A	18	U30 USB driver	400*500
SG_DEV_FDU10A	19	U-AIR USB driver	500*700
SG_DEV_FDUSDA	13	U20-ASF-BT (Bluetooth SPP) driver	300*400
SG_DEV_FDUSDA_BLE	14	U20-ASF-BT (Bluetooth BLE) driver	300*400
SG_DEV_AUTO	255	Auto-detected	Depends on selected device

- **SGFPM_Init()**

```
DWORD devname = SG_DEV_AUTO;
err = SGFPM_Init(m_hFPM, devname);
```

- **SGFPM_InitEx2()**

```
DWORD image_width = 300;
DWORD image_height = 400;
DOWRD image_dpi = 500;
char* path_to_license_file;
err = SGFPM_InitEx2(m_hFPM, image_width, image_height, image_dpi,
path_to_license_file);
```

(SGFPM_InitEx() is no longer supported. It returns the error message: SGFDX_ERROR_NO_LONGER_SUPPORTED.)*

3.3. Terminating SGFPM

SGFPM_Terminate() must be called prior to terminating the application. It frees up the memory used by the SGFPM object.

```
if (m_hFPM)
{
    SGFPM_Terminate(m_hFPM);
}
m_hFPM = 0;
```


3.4. Opening the SecuGen Fingerprint Reader

To use a SecuGen fingerprint reader, call **SGFPM_OpenDevice()**. The second parameter (**devId**) of **SGFPM_OpenDevice()** can have different meanings depending on which type of fingerprint reader is used.

For USB readers, **devId** means device ID. If only one USB fingerprint reader is connected to the PC, **devId** will be 0. If multiple USB fingerprint readers are connected to one PC, **devId** can range from 0 to 9. The maximum number of SecuGen USB readers that can be connected to one PC is 10.

If **devId** is 0 (**AUTO_DETECT**), the device driver will find the port address automatically. The port address is defined in `sgfplib.h`.

In general, if only one USB reader is connected to the PC, then **0** or **USB_AUTO_DETECT**¹ is recommended.

```
DWORD devId = USB_AUTO_DETECT;           // auto detect
err = SGFPM_OpenDevice(m_hFPM, devId);
```

For U20-ASF-BT (BLE) devices (**SG_DEV_FDUSDA_BLE**), **SGFPM_OpenDevice2()** should be called with a device name from **SGFPM_FindDevices()** or in wide character string format² Instead.

```
DWORD ndevs = 0;
SGDeviceInfo *devList = NULL;
FindDevices(m_hFPM, &ndevs, &devList); // calls SGFPM_FindDevices()
if (ndevs > 0) {
    err = SGFPM_OpenDevice2(m_hFPM, devList[0].ID);
} else {
    err = SGFDX_ERROR_DEVICE_NOT_FOUND;
}
```

3.5. Getting Device Information

Device information can be retrieved by calling **SGFPM_GetDeviceInfo()**, which obtains required device information such as image height and width. The device information is contained in the **SGDeviceInfoParam** structure. Refer to [Chapter 5. Structure Reference](#) for a detailed description of the **SGDeviceInfoParam** structure.

```
SGDeviceInfoParam device_info;
memset(&device_info, 0x00, sizeof(device_info));
error = SGFPM_GetDeviceInfo(m_hFPM, &device_info);

if (error == SGSGFDX_ERROR_NONE)
{
    m_DevID = device_info.DeviceID;
    m_DevSN = device_info.DevicesN;
    m_ImgWidth = device_info.ImageWidth;
    m_ImgHeight = device_info.ImageHeight;
    m_Contrast = device_info.Contrast;
```

¹ For U20-ASF-BT, it should be a com port. See the sample code for more information.

² For example, a U20-ASF-BT (BLE) device name looks like L"BluetoothLE#BluetoothLEac:d1:b8:d0:d6:e4-cc:35:5a:ff:f0:37"

```

    m_Brightness = device_info.Brightness;
    m_Gain = device_info.Gain;
    m_ImageDPI = device_info.ImageDPI;
    char buffer[20];
    _ultoa(device_info.FWVersion, buffer, 16);
    m_FWVersion = CString(buffer);
}

```

For U20-ASF-BT (BLE) devices, **SGFPM_FindDevices()** can be used to enumerate them.

```

DWORD ndevs = 0;
SGDeviceInfo *devList = NULL;
int timeout = 40000; // 40 seconds

// finding devices for timeout in milliseconds or less
DWORD err = SGFPM_FindDevices(m_hFPM, &ndevs, &devList, timeout);

```

To cancel finding U20-ASF-BT (BLE) devices, **SGFPM_CancelFind()** can be called in the other thread.

3.6. Capturing a Fingerprint Image

After the reader is initialized, a fingerprint image can be captured. The SGFPM object provides three types of fingerprint image capture functions listed below. Captured fingerprints are 256 gray-level images, and image width and height can be retrieved by calling **SGFPM_GetDeviceInfo()**. The image buffer should be allocated by the calling application.

SGFPM_GetImage() captures an image without checking for the presence of a finger or checking image quality. **SGFPM_GetImageEx()** captures fingerprint images continuously, checks the image quality against a specified quality value and ignores the image if it does not contain a fingerprint or if the quality of the fingerprint is not acceptable. If a quality image is captured within the given time (the second parameter), **SGFPM_GetImageEx()** ends its processing. If a window handle is provided by the application, the drivers will draw a fingerprint image in the provided window using the handle value. For more information about each of the following SGFPM image capture functions, refer to [Chapter 4. Function Reference](#).

If **SGFPM_GetImage()** needs to be called multiple times, it is recommended to call **SGFPM_BeginGetImage()** before and **SGFPM_EndGetImage()** after the series of calls for **SGFPM_GetImage()**. This will allow **SGFPM_GetImage()** to run faster if the device, such as U-Air, supports these two functions. For more information, please review the MatchingUAIR sample code.

- **SGFPM_GetImage()**

```

[Example]
BYTE *buffer = new BYTE(m_ImageWidth*m_ImageHeight);

if (SGFPM_GetImage(m_hFPM, buffer) == SGSGFDX_ERROR_NONE) // Get image data from
device
{
    // Display image
    // Process image
}
delete [] buffer;

```

- **SGFPM_GetImageEx()**

```

DWORD timeout = 10000;
DWORD quality = 80;
if(SGFPM_GetImageEx(m_hFPM, buffer, timeout, NULL, quality) == SGFDX_ERROR_NONE)
{
    // Draw image
}

```

For U20-ASF-BT (BLE) devices, **SGFPM_CreateTemplateDev()** and **SGFPM_GetTemplateDev()** are recommended because transferring images wirelessly can take a long time, which may not be practical.

```

// capture and create a template
DWORD template_size = 0;
dwErr = SGFPM_CreateTemplateDev(hFPM, &template_size);
assert(dwErr == SGFDX_ERROR_NONE);
assert(template_size > 0);

// get template data
if (template_size > 0) {
    char *template_data = new char[template_size];
    dwErr = SGFPM_GetTemplateDev(hFPM, (BYTE*)template_data);
    assert(dwErr == SGFDX_ERROR_NONE);

    delete[] template_data;
}

```

The template format can be retrieved from a U20-ASF-BT (BLE) device by calling **SGFPM_GetTemplateFormatDev()**. The format can be set by calling **SGFPM_SetTemplateFormatDev()**. These work only with **SGFPM_CreateTemplateDev()**.

```

// get the current template format
WORD current_template_format = 0;
dwErr = SGFPM_GetTemplateFormatDev(hFPM, &current_template_format);
assert(dwErr == SGFDX_ERROR_NONE);

// set template
WORD template_format = format;
dwErr = SGFPM_SetTemplateFormatDev(hFPM, template_format);
assert(dwErr == SGFDX_ERROR_NONE);

```

3.7. Getting Image Quality

To determine the fingerprint image quality, use **SGFPM_GetImageQuality()**. For a simple and fast way to determine the quality of the last image captured from the device, use **SGFPM_GetLastImageQuality()**³.

- **SGFPM_GetImageQuality()**

```
DWORD img_qlty;
SGFPM_GetImageQuality(hFPM, ImageWidth, m_ImageHeight, fp_image, &mg_qlty);
if (img_qlty < 80)
    // Capture again
```

3.8. Controlling Brightness

Depending on the fingerprint reader used, environmental factors and the specifications of the host system, the brightness of a fingerprint image may vary. To improve the quality of a captured image, the image brightness should be adjusted by changing the brightness setting of the reader using **SGFPM_Configure()** or **SGFPM_SetBrightness()**. Using **SGFPM_Configure()** presents a built-in dialog box in the driver from which the user can easily adjust brightness and receive instant feedback from the fingerprint image displayed. **SGFPM_SetBrightness()** can also be used to control brightness of the reader. Brightness default values vary among the different types of SecuGen readers.

- **SGFPM_Configure()**

```
HWND hwnd = 0;
SGFPM_SetBrightness(m_hFPM, hwnd);          // Show device configuration box in the
driver.
```

- **SGFPM_SetBrightness()**

```
SGFPM_SetBrightness(m_hFPM, 70); // Set from 0 to 100.
```

3.9. Creating a Template

To register or verify a fingerprint, a fingerprint image is first captured, and then feature data (minutiae) is extracted from the image into a **template**. Minutiae are the unique core points near the center of every fingerprint, such as ridges, ridge endings, bifurcations, valleys and whorls.

Use **SGFPM_CreateTemplate()** to extract minutiae from a fingerprint image to form a template. The buffer should be assigned by the application. To get the buffer size of the minutiae, call **SGFPM_GetMaxTemplateSize()**. It will return the maximum buffer size for data in one template. The actual template size can be obtained by calling **SGFPM_GetTemplateSize()** after the template is created. The **SGFPM_CreateTemplate()** API creates only one set of data from an image.

Note: Templates having the ANSI378, ISO19794-2, or ISO19794-2 compact card formats may be merged. For more information about template formats and merging formats, refer to the following Sections:

[Section 3.15. Template Format](#)

[Section 3.16. Manipulating ANSI378 Templates](#)

³ It depends on the device. Not all devices support this function. The return value should be checked.

[Section 3.17. Manipulating ISO19794-2 Templates](#)

[Section 3.18. Manipulating ISO19794-2 Compact Card Templates](#)

- **SGFPM_CreateTemplate()**

```
// Get a fingerprint image
DWORD qlty = 80;
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);

// Create template from captured image
BYTE* minBuffer;
err = SGFPM_GetMaxTemplateSize(m_hFPM, &maxTemplateSize);
minBuffer = new BYTE[maxTemplateSize];

// Set information about template
SGFingerInfo finger_info;
finger_info.FingerNumber = GetFingerPos();
finger_info.ImageQuality = (WORD)qlty;
finger_info.ImpressionType = SG_IMPTYPE_LP;
finger_info.ViewNumber = 0;

err = SGFPM_CreateTemplate(m_hFPM, &finger_info, m_ImgBuf, minBuffer);
```

3.10. Matching Templates

Templates are matched during both registration and verification processes. During registration, it is recommended to capture at least two image samples per fingerprint for a higher degree of accuracy. The minutiae data from each image sample can then be compared against each other (i.e. matched) to confirm the quality of the registered fingerprints. This comparison is analogous to a password confirmation routine that is commonly required for entering a new password.

During verification, newly input minutiae data is compared against registered minutiae data. Similar to the registration process, verification requires the capture of a fingerprint image followed by extraction of the minutiae data from the captured image into a template.

To match templates, FDx SDK Pro provides four kinds of matching functions. Each function requires two sets of template data for matching.

SGFPM_MatchTemplate(): This function matches templates having the same format as the default format. When calling this function, each template should include only one sample (or view) per template. The default format is SG400 (SecuGen proprietary format) but can be changed by calling `SGFPM_SetTemplateFormat()`. For more information about template formats, refer to [Section 3.15. Template Format](#).

SGFPM_MatchTemplateEx(): This function can match templates having different template formats. This function can also specify the template format for each template and can match templates that have multiple views per template.

SGFPM_MatchAnsiTemplate(): This function is the same as **SGFPM_MatchTemplateEx()** except that it supports only ANSI378 templates.

SGFPM_MatchIsoTemplate(): This function is the same as **SGFPM_MatchTemplateEx()** except that it

supports only ISO19794-2 templates.

SGFPM_MatchIsoCompactTemplate(): This function is the same as **SGFPM_MatchTemplateEx()** except that it supports only ISO19794-2 compact card templates.

Function	Template Format	Can match templates with different formats?
SGFPM_MatchTemplate	SG400 (System default)	No
SGFPM_MatchTemplateEx	Specified template format	Yes
SGFPM_MatchAnsiTemplate	ANSI378	No
SGFPM_MatchIsoTemplate	ISO19794-2	No
SGFPM_MatchIsoCompactTemplate	ISO19794-2 compact card template	No

- **SGFPM_MatchTemplate()**

```

BYTE*   m_RegTemplate1;
BYTE*   m_RegTemplate2;
...
// Getfirst fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate1);

// Get second fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate2);

DWORD sl = SL_NORMAL;           // Set security level as NORMAL
BOOL matched;
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplate1, m_RegTemplate2, sl, &matched);

```

- **SGFPM_MatchTemplateEx()**

```

BYTE*   m_RegTemplate1;    // Will contain SG400 template
BYTE*   m_RegTemplate2;    // Will contain ANSI378 template
...
// Make SG400 template
err = SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_SG400);
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate1);

// Make ANSI378 template
err = SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ANSI378);
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate2);

DWORD sl = SL_NORMAL;           // Set security level as NORMAL
BOOL matched;
err = SGFPM_MatchTemplateEx(m_hFPM, m_RegTemplate1,
                           TEMPLATE_FORMAT_SG400,
                           0, // Must be 0 if template format is SG400
                           m_RegTemplate2,
                           TEMPLATE_FORMAT_ANSI378,

```

```
0,    // Currently only one sample
sl,
&matched);
```

- **SGFPM_MatchAnsiTemplate()**

```
DWORD err;
BOOL matched = FALSE;
SGANSITemplateInfo sample_info;
err = SGFPM_GetAnsiTemplateInfo(m_hFPM, m_EnrollTemplate, &sample_info);

matched = TRUE;
bool finger_found = false;
for (int i = 0; i < sample_info.TotalSamples; i++)
{
    if(sample_info.SampleInfo[i].FingerNumber == finger_pos) // Try match for same
    finger
    {
        finger_found = true;
        err = SGFPM_MatchAnsiTemplate(m_hFPM, m_EnrollTemplate, i, m_FetBufM, 0,
        SecurityLevel[m_SecureLevel.GetCurSel()], &matched);
        if (!matched)
            break;
    }
}
```

- **SGFPM_MatchIsoTemplate()**

```
DWORD err;
BOOL matched = FALSE;

// ISO19794-2
SGISOTemplateInfo sample_info = {0};
err = SGFPM_GetIsoTemplateInfo(m_hFPM, m_StoredTemplate, &sample_info);

matched = FALSE;
int found_finger = -1;
for (int i = 0; i < sample_info.TotalSamples; i++)
{
    // ISO19794-2
    err = SGFPM_MatchIsoTemplate(m_hFPM, m_StoredTemplate, i, m_FetBufM, 0,
    SL_NORMAL, &matched);
    if (matched)
    {
        found_finger = sample_info.SampleInfo[i].FingerNumber;
        break;
    }
}
```

- **SGFPM_MatchIsoCompactTemplate()**

```
DWORD err;
BOOL matched = FALSE;

// ISO19794-2
SGISOTemplateInfo sample_info = {0};
err = SGFPM_GetIsoCompactTemplateInfo(m_hFPM, m_StoredTemplate, &sample_info);
```

```

matched = FALSE;
int found_finger = -1;
for (int i = 0; i < sample_info.TotalSamples; i++)
{
    // ISO19794-2 compact card template
    err = SGFPM_MatchIsoCompactTemplate(m_hFPM, m_StoredTemplate, i,
    m_FetBufM, 0, SL_NORMAL, &matched);
    if (matched)
    {
        found_finger = sample_info.SampleInfo[i].FingerNumber;
        break;
    }
}

```

3.11. Registration process

To register a fingerprint, a fingerprint image is first captured, and then feature data (minutiae) is extracted from the image into a template. It is recommended to capture at least two image samples per fingerprint for a higher degree of accuracy. The minutiae data from each image can then be compared against each other (i.e. matched) to confirm the quality of the registered fingerprints. This comparison is analogous to a password confirmation routine that is commonly required for entering a new password.

Overview of Registration Process

1. Capture fingerprint images: **SGFPM_GetImage()** or **SGFPM_GetImageEx()**
2. Extract minutiae from each captured fingerprint image: **SGFPM_CreateTemplate()**
3. Match each template to determine if they are acceptable for registration: **SGFPM_MatchTemplate()**
4. Save templates to file or database to complete registration

Example: Using two fingerprint images to register one fingerprint

```

BYTE*   m_RegTemplate1;
BYTE*   m_RegTemplate2;

err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_RegTemplate1 = new BYTE[m_MaxTemplateSize];
m_RegTemplate2 = new BYTE[m_MaxTemplateSize];

// Get first fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate1);

// Get second fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_RegTemplate2);

DWORD sl = SL_NORMAL; // Set security level as NORMAL
BOOL matched;
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplate1, m_RegTemplate2, sl, &matched);

if (matched)
    // Save these templates somewhere

```


3.12. Verification Process

The verification process involves matching newly input minutiae data against registered minutiae data. Similar to the registration process, verification requires the capture of a fingerprint image followed by extraction of the minutiae data from the captured image into a template.

Overview of Verification Process

1. Capture fingerprint image: **SGFPM_GetImage()** or **SGFPM_GetImageEx()**
2. Extract minutiae data from captured image: **SGFPM_CreateTemplate()**
3. Match newly made template against registered templates: **SGFPM_MatchTemplate()**

- Adjust the security level according to the type of application. For example, if fingerprint-only authentication is used, set the security level higher than **SL_NORMAL** to reduce false acceptance (FAR).

Example: Input minutiae data is matched against two registered minutiae data samples

```
BYTE* m_VrfTemplate1;

err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_VrfTemplate1= new BYTE[m_MaxTemplateSize];

// Get first fingerprint image and create template from the image
DWORD qlty = 50;
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, qlty);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_VrfTemplate1);

DWORD sl = SL_NORMAL; // Set security level depending on applications.
DWORD err;
BOOL matched1, matched2;
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplate1, m_VrfTemplate1, sl);
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplate2, m_VrfTemplate1, sl);

if (err == SGSGFDX_ERROR_NONE)
{
    if (matched1 && matched2)
        // Matched
    else
        // Not matched
}
```

3.13. Getting Matching Score

For improved quality control during the registration or verification process, a matching score can be used instead of a security level setting to determine the success of the operation. The matching score can be specified so that only sets of minutiae data that exceed the score will be accepted; data below the score will be rejected. The matching score may have a value from 0 to 199. **SGFPM_GetMatchingScore()** requires two sets of minutiae data of the same template format. **SGFPM_GetMatchingScoreEx()** requires two sets of minutiae data, but they can take different template formats. For more information about template formats, refer to [Section 3.15. Template Format](#).

For more information about **SGFPM_GetMatchingScoreEx()**, refer to [Section 4.5. Matching Functions](#).

```

DWORD score;
if (SGFPM_GetMatchingScore(m_hFPM, m_RegTemplat1, m_RegTemplat1, &score) ==
SGSGFDX_ERROR_NONE)
{
    if (score > 100)
        // Enroll these fingerprints to database
    else
        // Try again
}

```

To understand how the matching score correlates with typical security levels, refer to the chart below. For more information about security levels, refer to [Section 4.5. Matching Functions](#).

Security Level vs. Matching Score

Security Level	Value	Matching Score
SL_NONE	0	0
SL_LOWEST	1	30
SL_LOWER	2	50
SL_LOW	3	60
SL_BELOW_NORMAL	4	70
SL_NORMAL	5	80
SL_ABOVE_NORMAL	6	90
SL_HIGH	7	100
SL_HIGHER	8	120
SL_HIGHEST	9	140

Note: The Matching Scores have changed after version 3.53 of FDX SDK Pro.

3.14. Using Auto-On™

Auto-On™ is a function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. To use this function, Auto-On should be enabled using **SGFPM_EnableAutoOnEvent()**. Once Auto-On is enabled, the application can receive a message from the device driver whenever an Auto-On event occurs in the reader.

When calling **SGFPM_EnableAutoOnEvent()**, pass the handle of the window which will receive the Auto-On message. The Auto-On message is defined as 0x8100 in sgplib.h. When the application receives an Auto-On message, wParam will have event type (Finger ON or OFF) and lParam will have information of the device from which the event occurred.

Note: Auto-On is not supported by FDU02-based readers.

- **Enabling Auto-On**

```

[Example]
#define WM_APP_SGAUTOONEVENT    0x8100

if ((device_name == SG_DEV_FDU03) || (device_name == SG_DEV_FDU04))

```

```
{
    SGFPM_EnableAutoOnEvent(m_hFPM, TRUE, this.m_hWnd, 0);
}
```

- **Disabling Auto-On**

```
[Example]
if ((device_name == SG_DEV_FDU03) || (device_name == SG_DEV_FDU04))
{
    SGFPM_EnableAutoOnEvent(m_hFPM, FALSE, this.m_hWnd, 0);
}
```

- **Handling Auto-On event in application**

```
ON_MESSAGE (WM_APP_SGAUTOONEVENT, OnAutoOnEvent)
...
LRESULT CSgdvcDlg::OnAutoOnEvent(WPARAM wParam, LPARAM lParam)
{
    WORD isfinger = wParam;
    SGDeviceInfoParam device_info;
    memcpy(&device_info, (SGDeviceInfoParam*)lParam, sizeof(device_info));

    if(isfinger ==SGDEVEVNET_FINGER_ON)
    {
        m_ErrorDisplay.Format(_T("Device Event:Finger ON, DevId:%d, SN:%s"),
            device_info.DeviceID, device_info.DeviceSN);
        DWORD err = m_Sensor->GetImageEx(m_ImgBuf, 3000, m_ImageBox.m_hWnd, 30);
    }
    else if(isfinger == SGDEVEVNET_FINGER_OFF)
    {
        m_ErrorDisplay = TEXT("Device event: Finger Off");
    }
    UpdateData(FALSE);
    return 1;
}
```

3.15. Template Format

The FDx SDK Pro supports four types of fingerprint template formats:

- **SG400:** SecuGen's proprietary template format
- **ANSI378:** ANSI-INCITS 378-2004 "Finger Minutiae Format for Data Exchange"
- **ISO19794-2:** ISO/IEC 19794-2:2005 "Biometric Data Interchange Formats – Part 2: Finger Minutiae Data"
- **ISO19794-2 Compact⁴:** ISO/IEC 19794-2:2005 "Biometric Data Interchange Formats– Part 2: Finger Minutiae Data" – Section 8.2 Compact Size Finger Minutiae Format (Compact Card Format)

As default, SGFPM creates SecuGen proprietary templates (TEMPLATE_FORMAT_SG400). To change the template format, use **SGFPM_SetTemplateFormat()**.

⁴ U20-ASF-BT (Bluetooth SPP and BLE) devices do not support this format.

SG400 templates are encrypted for high security and have a size of 400 bytes. ANSI378 and ISO19794-2 templates are not encrypted, and their size is variable depending on how many fingers are registered in the structure and how many minutiae points are found.

For more information about the ANSI378 template, refer to the standard document titled “Information technology - Finger Minutiae Format for Data Interchange,” (document number ANSI-INCITS 378-2004) available at the ANSI website <http://webstore.ansi.org>.

For more information about the ISO19794-2 and ISO19794-2 Compact templates, refer to the standard document titled “Information technology--Biometric Data Interchange Formats--Part 2: Finger Minutiae Data,” (document number ISO / IEC 19794-2:2005) available at the ISO website <https://www.iso.org/standard/38746.html>.

Once the template format is set, it will affect the execution of the SGFPM module.

The following APIs are affected by **SGFPM_SetTemplateFormat()**:

- **SGFPM_GetMaxTemplateSize()**
- **SGFPM_CreateTemplate()**
- **SGFPM_GetTemplateSize()**
- **SGFPM_MatchTemplate()**
- **SGFPM_GetMatchingScore()**

The following APIs work only when the template format is **TEMPLATE_FORMAT_ANSI378**:

- **SGFPM_GetTemplateSizeAfterMerge()**
- **SGFPM_MergeAnsiTemplate()**
- **SGFPM_MergeMultipleAnsiTemplate()**
- **SGFPM_GetAnsiTemplateInfo()**
- **SGFPM_MatchAnsiTemplate()**
- **SGFPM_GetAnsiMatchingScore()**

The following APIs work only when the template format is **TEMPLATE_FORMAT_ISO19794**:

- **SGFPM_GetIsoTemplateSizeAfterMerge()**
- **SGFPM_MergeIsoTemplate()**
- **SGFPM_MergeMultipleIsoTemplate()**
- **SGFPM_GetIsoTemplateInfo()**
- **SGFPM_MatchIsoTemplate()**
- **SGFPM_GetIsoMatchingScore()**

The following APIs work only when the template format is **TEMPLATE_FORMAT_ISO19794_COMPACT**:

- **SGFPM_GetIsoCompactTemplateSizeAfterMerge()**
- **SGFPM_MergeIsoCompactTemplate()**
- **SGFPM_MergeMultipleIsoCompactTemplate()**
- **SGFPM_GetIsoCompactTemplateInfo()**
- **SGFPM_MatchIsoCompactTemplate()**
- **SGFPM_GetIsoCompactMatchingScore()**

The following APIs work with any template format:

- **SGFPM_MatchTemplateEx()**
- **SGFPM_GetMatchingScoreEx()**

The following APIs work for Bluetooth SPP (U20-ASF-BT, SPP) devices and do not affect the other APIs above.

- **SGFPM_CreateTemplateDev()**
- **SGFPM_GetTemplateDev()**

- **SGFPM_GetTemplateFormatDev()**
- **SGFPM_SetTemplateFormatDev()**

- **Defining template format**

```
enum SGFDxTemplateFormat
{
    TEMPLATE_FORMAT_ANSI378 = 0x0100,
    TEMPLATE_FORMAT_SG400 = 0x0200,
    TEMPLATE_FORMAT_ISO19794 = 0x0300,
    TEMPLATE_FORMAT_ISO19794_COMPACT = 0x0400,
};
```

- **Setting template format to ANSI378**

```
SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ANSI378);
```

- **Setting template format to SG400**

```
SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_SG400);
```

- **Setting template format to ISO19794**

```
SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ISO19794);
```

- **Setting template format to ISO19794 Compact Card Format**

```
SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ISO19794_COMPACT);
```

3.16. Manipulating ANSI378 Templates

The ANSI378 template format allows multiple fingers and multiple views per finger to be stored in one template. To support this feature, FDx SDK Pro provides the following special APIs:

- **SGFPM_GetTemplateSizeAfterMerge()**
- **SGFPM_MergeAnsiTemplate()**
- **SGFPM_MergeMultipleAnsiTemplate()**
- **SGFPM_GetAnsiTemplateInfo()**
- **SGFPM_MatchAnsiTemplate()**
- **SGFPM_GetAnsiMatchingScore()**

- **Merging two ANSI378 templates**

After creating an ANSI378 template from a fingerprint image, additional ANSI378 templates can be merged into one template. To do this, use **SGFPM_MergeAnsiTemplate()**, which takes two ANSI378 templates and merges them into one template. The size of the merged template will be smaller than the sum of the sizes of all input templates. Call **SGFPM_GetTemplateSizeAfterMerge()** to obtain the exact template size of the merged template before using **SGFPM_MergeAnsiTemplate()**.

```
BYTE*    m_Template1;
BYTE*    m_Template2;
```

```

err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_Template1 = new BYTE[m_MaxTemplateSize];
m_Template2 = new BYTE[m_MaxTemplateSize];

// Get first fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template1);

// Get second fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template2);

// Save template after merging two templates - m_Template1, m_Template2
BYTE* merged_template;
DWORD buf_size;

err = SGFPM_GetTemplateSizeAfterMerge(m_hFPM, m_Template1, m_Template2,
&buf_size);
merged_template = new BYTE[buf_size];
err = SGFPM_MergeAnsiTemplate(m_hFPM, m_Template1, m_Template2, merged_template);

// Save m_EnrollTemplate to file
...
SaveTemplate(file_name, merged_template, buf_size);
delete [] merged_template; // Freed by calling application

```

- **Merging multiple ANSI378 templates**

More than two ANSI378 templates may be merged into one template using **SGFPM_MergeMultipleAnsiTemplate()**. The size of the merged template will be smaller than the sum of the sizes of all source templates. To determine the buffer size for the merged template, use the sum of the size for each template and then later obtain the actual size of merged template after calling **SGFPM_MergeMultipleAnsiTemplate()**.

```

BYTE* target_template;
DWORD size1, size2;
DWORD err;
DWORD real_size = 0;

// Buffer for input templates - source template
err = SGFPM_GetTemplateSize(m_hFPM, template1, &size1);
err = SGFPM_GetTemplateSize(m_hFPM, template2, &size2);

BYTE* source_template = new BYTE[size1 + size2]; // Make stack of each template
memcpy(&source_template[0], template1, size1);
memcpy(&source_template[size1], template2, size2);

// Allocate buffer for output template - merged template
target_template = new BYTE[size1+ size2];

err = SGFPM_MergeMultipleAnsiTemplate(m_hFPM, source_template, 2,
target_template);
delete [] source_template;

// Get actual size of merged_template

```

```
// Actual size will be less than size1+size2
err = SGFPM_GetTemplateSize(m_hFPM, target_template, &real_size);
```

- **Getting information about an ANSI378 template**

The ANSI378 template format allows multiple fingers and multiple views per finger to be stored in one template. To match one sample (view) against a sample in other template, information about the template may be needed. To get sample information about a template, use **SGFPM_GetAnsiTemplateInfo()**.

```
DWORD err;
int matched_samples = 0;

SGANSITemplateInfo sample_info1, sample_info2;
err = SGFPM_GetAnsiTemplateInfo(m_hFPM, g_EnrollData, &sample_info1);
err = SGFPM_GetAnsiTemplateInfo(m_hFPM, g_VrfData, &sample_info2);

for (int i = 0; i < sample_info1.TotalSamples; i++)
{
    for (int j = 0; j < sample_info2.TotalSamples; j++)
    {
        BOOL matched;
        err = SGFPM_MatchAnsiTemplate(m_hFPM, g_EnrollData, i, g_VrfData, 0, sl,
&matched);
        if (matched)
            matched_samples++;
    }
}

if (err == SGFDX_ERROR_NONE)
{
    if (matched_samples)
        m_ResultEdit.Format("Found %d matched samples: ", matched_samples);
    else
        m_ResultEdit.Format("Cannot find matched sample");
}
else
    m_ResultEdit.Format("MatchTemplate() failed. Error = %d ", err);
```

3.17. Manipulating ISO19794-2 Templates

The ISO19794-2 template format allows multiple fingers and multiple views per finger to be stored in one template. To support this feature, FDx SDK Pro provides the following special APIs:

- **SGFPM_GetIsoTemplateSizeAfterMerge()**
- **SGFPM_MergelsoTemplate()**
- **SGFPM_MergeMultipleIsoTemplate()**
- **SGFPM_GetIsoTemplateInfo()**
- **SGFPM_MatchIsoTemplate()**
- **SGFPM_GetIsoMatchingScore()**

- **Merging two ISO19794-2 templates**

After creating an ISO19794-2 template from a fingerprint image, additional ISO19794-2 templates can be merged into one template. To do this, use **SGFPM_MergelsoTemplate()**, which takes two ISO19794-2

templates and merges them into one template. The size of the merged template will be smaller than the sum of the sizes of all input templates. Call **SGFPM_GetIsoTemplateSizeAfterMerge()** to obtain the exact template size of the merged template before using **SGFPM_MergeIsoTemplate()**.

```

BYTE*   m_Template1;
BYTE*   m_Template2;

// Set template format to ISO19794-2
err = SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ISO19794);

err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_Template1 = new BYTE[m_MaxTemplateSize];
m_Template2 = new BYTE[m_MaxTemplateSize];

// Get first fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template1);

// Get second fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template2);

// Save template after merging two templates - m_Template1, m_Template2
BYTE*   merged_template;
DWORD   buf_size;

err = SGFPM_GetIsoTemplateSizeAfterMerge(m_hFPM, m_Template1, m_Template2,
&buf_size);
merged_template = new BYTE[buf_size];
err = SGFPM_MergeIsoTemplate(m_hFPM, m_Template1, m_Template2, merged_template);

// Save m_EnrollTemplate to file
...
SaveTemplate(file_name, merged_template, buf_size);
delete [] merged_template; // Freed by calling application

```

- **Merging multiple ISO19794-2 templates**

More than two ISO19794-2 templates may be merged into one template using **SGFPM_MergeMultipleIsoTemplate()**. The size of the merged template will be smaller than the sum of the sizes of all source templates. To determine the buffer size for the merged template, use the sum of the size for each template and then later obtain the actual size of merged template after calling **SGFPM_MergeMultipleIsoTemplate()**.

```

BYTE*   target_template;
DWORD   size1, size2;
DWORD   err;
DWORD   real_size = 0;

// Buffer for input templates - source template
err = SGFPM_GetTemplateSize(m_hFPM, template1, &size1);
err = SGFPM_GetTemplateSize(m_hFPM, template2, &size2);

BYTE*   source_template = new BYTE[size1 + size2]; // Make stack of each template
memcpy(&source_template[0], template1, size1);

```



```

memcpy(&source_template[size1], template2, size2);

// Allocate buffer for output template - merged template
target_template = new BYTE[size1+ size2];

err = SGFPM_MergeMultipleIsoTemplate(m_hFPM, source_template, 2, target_template);
delete [] source_template;

// Get actual size of merged_template
// Actual size will be less than size1+size2
err = SGFPM_GetTemplateSize(m_hFPM, target_template, &real_size);

```

- **Getting information about an ISO19794-2 template**

The ISO19794-2 template format allows multiple fingers and multiple views per finger to be stored in one template. To match one sample (view) against a sample in other template, information about the template may be needed. To get sample information about a template, use **SGFPM_GetIsoTemplateInfo()**.

```

DWORD err;
BOOL matched = FALSE;

// ISO19794-2
SGISOTemplateInfo sample_info = {0};
err = SGFPM_GetIsoTemplateInfo(m_hFPM, m_StoredTemplate, &sample_info);

matched = FALSE;
int found_finger = -1;
for (int i = 0; i < sample_info.TotalSamples; i++)
{
    // ISO19794-2
    err = SGFPM_MatchIsoTemplate(m_hFPM, m_StoredTemplate, i, m_FetBufM, 0,
    SL_NORMAL,
                                &matched);

    if (matched)
    {
        found_finger = sample_info.SampleInfo[i].FingerNumber;
        break;
    }
}

if (err == SGFDX_ERROR_NONE)
{
    if (found_finger >= 0)
        m_ResultEdit.Format("The fingerprint data found. Finger Position: %s",
                             g_FingerPosStr[found_finger]);
    else
        m_ResultEdit.Format("Cannot find matched fingerprint data");
}
else
{
    m_ResultEdit.Format("MatchIsoTemplate() failed. Error = %d ", err);
}

```

3.18. Manipulating ISO19794-2 Compact Card Templates

The ISO19794-2 compact card template format allows multiple fingers and multiple views per finger to be stored in one template. It also encodes minutiae in the compact size finger minutiae format. To support this feature, FDx SDK Pro provides the following special APIs:

- **SGFPM_GetIsoTemplateSizeAfterMerge()**
- **SGFPM_MergelsoTemplate()**
- **SGFPM_MergeMultipleIsoTemplate()**
- **SGFPM_GetIsoTemplateInfo()**
- **SGFPM_MatchIsoTemplate()**
- **SGFPM_GetIsoMatchingScore()**

- **Merging two ISO19794-2 compact card templates**

After creating an ISO19794-2 compact card template from a fingerprint image, additional ISO19794-2 compact card templates can be merged into one template. To do this, use **SGFPM_MergelsoCompactTemplate()**, which takes two ISO19794-2 compact card templates and merges them into one template. The size of the merged template will be smaller than the sum of the sizes of all input templates. Call **SGFPM_GetIsoCompactTemplateSizeAfterMerge()** to obtain the exact template size of the merged template before using **SGFPM_MergelsoCompactTemplate()**.

```

BYTE*   m_Template1;
BYTE*   m_Template2;

// Set template format to ISO19794-2 compact card format
err = SGFPM_SetTemplateFormat(m_hFPM, TEMPLATE_FORMAT_ISO19794_COMPACT);

err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_Template1 = new BYTE[m_MaxTemplateSize];
m_Template2 = new BYTE[m_MaxTemplateSize];

// Get first fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template1);

// Get second fingerprint image and create template from the image
err = SGFPM_GetImageEx(m_hFPM, m_ImgBuf, 5000, NULL, 80);
err = SGFPM_CreateTemplate(m_hFPM, 0, m_ImgBuf, m_Template2);

// Save template after merging two templates - m_Template1, m_Template2
BYTE*   merged_template;
DWORD   buf_size;

err = SGFPM_GetIsoCompactTemplateSizeAfterMerge(m_hFPM, m_Template1, m_Template2,
&buf_size);
merged_template = new BYTE[buf_size];
err = SGFPM_MergeIsoCompactTemplate(m_hFPM, m_Template1, m_Template2,
merged_template);

// Save m_EnrollTemplate to file
...
SaveTemplate(file_name, merged_template, buf_size);
delete [] merged_template; // Freed by calling application

```

- **Merging multiple ISO19794-2 compact card templates**

More than two ISO19794-2 compact card templates may be merged into one template using **SGFPM_MergeMultipleIsoCompactTemplate()**. The size of the merged template will be smaller than the sum of the sizes of all source templates. To determine the buffer size for the merged template, use the sum of the size for each template and then later obtain the actual size of merged template after calling

SGFPM_MergeMultipleIsoCompactTemplate().

```

BYTE* target_template;
DWORD size1, size2;
DWORD err;
DWORD real_size = 0;

// Buffer for input templates - source template
err = SGFPM_GetTemplateSize(m_hFPM, template1, &size1);
err = SGFPM_GetTemplateSize(m_hFPM, template2, &size2);

BYTE* source_template = new BYTE[size1 + size2]; // Make stack of each template
memcpy(&source_template[0], template1, size1);
memcpy(&source_template[size1], template2, size2);

// Allocate buffer for output template - merged template
target_template = new BYTE[size1+ size2];

err = SGFPM_MergeMultipleIsoCompactTemplate(m_hFPM, source_template, 2,
target_template);
delete [] source_template;

// Get actual size of merged_template
// Actual size will be less than size1+size2
err = SGFPM_GetTemplateSize(m_hFPM, target_template, &real_size);

```

- **Getting information about an ISO19794-2 compact card template**

The ISO19794-2 compact card template format allows multiple fingers and multiple views per finger to be stored in one template. To match one sample (view) against a sample in other template, information about the template may be needed. To get sample information about a template, use **SGFPM_GetIsoCompactTemplateInfo()**.

```

DWORD err;
BOOL matched = FALSE;

// ISO19794-2
SGISOTemplateInfo sample_info = {0};
err = SGFPM_GetIsoCompactTemplateInfo(m_hFPM, m_StoredTemplate, &sample_info);

matched = FALSE;
int found_finger = -1;
for (int i = 0; i < sample_info.TotalSamples; i++)
{
    // ISO19794-2
    err = SGFPM_MatchIsoCompactTemplate(m_hFPM, m_StoredTemplate, i, m_FetBufM,
0, SL_NORMAL,
                                &matched);

    if (matched)
    {
        found_finger = sample_info.SampleInfo[i].FingerNumber;
        break;
    }
}

if (err == SGFDX_ERROR_NONE)
{
    if (found_finger >= 0)
        m_ResultEdit.Format("The fingerprint data found. Finger Position: %s",

```

```

        g_FingerPosStr[found_finger]);
    else
        m_ResultEdit.Format("Cannot find matched fingerprint data");
}
else
{
    m_ResultEdit.Format("MatchIsoCompactTemplate() failed. Error = %d ", err);
}

```

3.19. Getting Version Information of MINEX Compliant Algorithms

To obtain version information about the MINEX Compliant algorithms, use **SGFPM_GetMinexVersion()**. Currently, the extractor version number is 0x000A0035, and the matcher version number is 0x000A8035.

```

DWORD extractor, matcher;
err = SGFPM_GetMinexVersion(m_hFPM, &extractor, &matcher);

CString sz_ver;
sz_ver.Format("(Extractor:0x%08X, Matcher:0x%08X)", extractor, matcher);
SetWindowText(_T("SecuGen ANSI MINEX Test ") + sz_ver);

```

Chapter 4. Function Reference

4.1. SGFPM Creation and Termination

DWORD SGFPM_Create(HSGFPM* hFPM)

Creates the SGFPM object internally

- **Parameters**
hFPM
 The pointer to contain the handle of the SGFPM object
- **Return values**
 SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_CREATION_FAILED = Failed to create SGFPM object

SGFPM_Terminate(HSGFPM hFpm)

Exits the SGFPM module

- **Parameters**
hFpm
 The handle of the SGFPM object
- **Return values**
 SGFDX_ERROR_NONE = No error

4.2. Initialization

DWORD SGFPM_Init(HSGFPM hFpm, DWORD devName)

Initializes SGFPM with device name information. The SGFPM object loads appropriate drivers with device name (devName) and initializes fingerprint algorithm module based on the device information.

- **Parameters**
hFpm
 The handle of the SGFPM object
devName
 Specifies the device name
 SG_DEV_FDU02: device name for FDU02-based USB readers
 SG_DEV_FDU03: device name for FDU03 and SDU03-based USB readers
 SG_DEV_FDU04: device name for FDU04 and SDU04-based USB readers
 SG_DEV_FDU05: device name for U20-based USB readers
 SG_DEV_FDU06: device name for UPx-based USB readers
 SG_DEV_FDU06AP: device name for UPx-AP based USB readers
 SG_DEV_FDU07: device name for U10-based USB readers
 SG_DEV_FDU08: device name for U20-A based USB readers
 SG_DEV_FDU08A: device name for U20-AP based USB readers
 SG_DEV_FDU08AL: device name for U20-AL based USB readers
 SG_DEV_FDU09A: device name for U30 based USB readers
 SG_DEV_FDU10A: device name for U-AIR based contactless USB readers
 SG_DEV_FDUSDA: device name for U20-ASF-BT (Bluetooth SPP) based readers
 SG_DEV_FDUSDA_BLE: device name for U20-ASF-BT (Bluetooth BLE) based readers

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_CREATION_FAILED = Failed to create SGFPM object
 SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 SGFDX_ERROR_DRVLOAD_FAILED = Failed to load driver

DWORD SGFPM_InitEx2(HSGFPM hFpm, DWORD width, DWORD height, DWORD dpi, char* licenseFilePath)

Initializes SGFPM with image information. Use when running fingerprint algorithm module without a SecuGen reader. Note: SGFPM_InitEx() is no longer supported.

- **Parameters**

hFpm

The handle of the SGFPM object

width

Image width in pixels

height

Image height in pixels

dpi

Image resolution in DPI

licenseFilePath

Path to a license file

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_CREATION_FAILED = Failed to create SGFPM object
 SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 SGFDX_ERROR_DLLLOAD_FAILED = Failed to load algorithm DLL
 SGFDX_ERROR_LICENSE_LOAD = Cannot find a license file
 SGFDX_ERROR_LICENSE_KEY = Invalid license key
 SGFDX_ERROR_LICENSE_EXPIRED = license expired

DWORD SGFPM_SetTemplateFormat(HSGFPM hFpm, WORD format)

Sets template format. Default format is SecuGen proprietary format (TEMPLATE_FORMAT_SG400).

- **Parameters**

hFpm

The handle of the SGFPM object

format

Specifies template format:

TEMPLATE_FORMAT_ANSI378: ANSI-INCITS 378-2004 format

TEMPLATE_FORMAT_SG400: SecuGen proprietary format

TEMPLATE_FORMAT_ISO19794: ISO/IEC 19794-2:2005 format

TEMPLATE_FORMAT_ISO19794_COMPACT: ISO/IEC 19794-2:2005 compact card format

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_CREATION_FAILED = Failed to create SGFPM object
 SGFDX_ERROR_INVALID_TEMPLATE_TYPE: Wrong template format

DWORD SGFPM_SetTemplateFormatDev(HSGFPM hFpm, WORD format)

Sets template format. Default format is SecuGen proprietary format (TEMPLATE_FORMAT_SG400). Only works with **SGFPM_CreateTemplateDev()**.

- **Parameters**

- hFpm***

- The handle of the SGFPM object

- format***

- Specifies template format:

- TEMPLATE_FORMAT_ANSI378: ANSI-INCITS 378-2004 format

- TEMPLATE_FORMAT_SG400: SecuGen proprietary format

- TEMPLATE_FORMAT_ISO19794: ISO/IEC 19794-2:2005 format

- The following format is not supported:

- TEMPLATE_FORMAT_ISO19794_COMPACT: ISO/IEC 19794-2:2005 compact card format

- **Return values**

- SGFDX_ERROR_NONE = No error

- SGFDX_ERROR_CREATION_FAILED = Failed to create SGFPM object

- SGFDX_ERROR_INVALID_TEMPLATE_TYPE: Wrong template format

4.3. Device and Capturing Functions

DWORD SGFPM_EnumerateDevice(HSGFPM hFpm, DWORD* ndevs, SGDeviceList devList, DWORD devName = SG_DEV_UNKNOWN)**

Enumerates currently attached reader to the system. If devName is not specified (SG_DEV_UNKNOWN), then it returns a list of all SecuGen readers attached to the system. If devName is specified, it enumerates only the device(s) that belong to the specified device class.

- **Parameters**

- hFpm***

- The handle of the SGFPM object

- ndevs***

- The number of attached USB readers

- devList***

- Buffer that contains device ID and device serial number. For more information, see [Section 5.3. SGDeviceList](#).

- devName***

- Device name:

- SG_DEV_UNKNOWN = 0

- SG_DEV_FDU02 = 0x03 // FDU02

- SG_DEV_FDU03 = 0x04 // FDU03 and SDU03

- SG_DEV_FDU04 = 0x05 // FDU04 and SDU04

- SG_DEV_FDU05 = 0x06 // U20

- SG_DEV_FDU06 = 0x07 // UPx

- SG_DEV_FDU06AP = 0x16 // UPx-AP

- SG_DEV_FDU07 = 0x08 // U10

- SG_DEV_FDU08 = 0x0A // U20-A

- SG_DEV_FDU08A = 0x11 // U20-AP

- SG_DEV_FDU08AL = 0x17 // U20-AL

- SG_DEV_FDU09A = 0x12 // U30

- SG_DEV_FDU10A = 0x13 // U-AIR

- SG_DEV_FDUSDA = 0x0D // U20-ASF-BT (Bluetooth SPP)

- SG_DEV_FDUSDA_BLE = 0x0E // U20-ASF-BT (Bluetooth BLE)

- **Returned values**

- SGFDX_ERROR_NONE = No error

- SGFDX_ERROR_FUNCTION_FAILED = General function fail error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

DWORD SGFPM_FindDevices(HSGFPM hFpm, DWORD* ndevs, SGDeviceList devList, DWORD timeout)**

Enumerates U20-ASF-BT (BLE) devices around the system. The last parameter, timeout, is recommended to be at least 10 seconds. Otherwise, ghost U20-ASF-BT (BLE) devices⁵ could be enumerated.

- **Parameters**

hFpm

The handle of the SGFPM object

ndevs

The number of U20-ASF-BT (BLE) devices found

devList

Buffer that contains device name and ID string. see [Section 5.3. SGDeviceList](#).

timeout

timeout in millisecond

- **Returned values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FUNCTION_FAILED = General function fail error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

DWORD SGFPM_CancelFind(HSGFPM hFpm)

Cancels finding U20-ASF-BT (BLE) devices.

- **Parameters**

hFpm

The handle of the SGFPM object

- **Returned values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FUNCTION_FAILED = General function fail error

DWORD SGFPM_OpenDevice(HSGFPM hFpm, DWORD devId)

Initializes the fingerprint reader.

- **Parameters**

hFpm

The handle of the SGFPM object

devId

Specifies the device ID for USB readers (value: 0 to 9). The maximum number of supported readers that may be attached at the same time is 10. If AUTO_DETECT is selected, the device driver will find its port address automatically.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_SYSLOAD_FAILED = Failed to loading system files

SGFDX_ERROR_INITIALIZE_FAILED = Failed to initialize chip

SGFDX_ERROR_DEVICE_NOT_FOUND = Device not found

⁵ A host system can have a list of U20-ASF-BT (BLE) devices, some of which are turned off. Those are removed from the list.

DWORD SGFPM_OpenDevice2(HSGFPM hFpm, wchar_t *devId)

Initializes the Bluetooth BLE fingerprint reader. (U20-ASF-BT BLE only)

- **Parameters**

- ***hFpm***

- The handle of the SGFPM object

- ***devId***

- Specifies the device ID string which can be retrieved by **SGFPM_FindDevices()**.

- **Return values**

- SGFDX_ERROR_NONE = No error

- SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

- SGFDX_ERROR_SYSLOAD_FAILED = Failed to loading system files

- SGFDX_ERROR_INITIALIZE_FAILED = Failed to initialize chip

- SGFDX_ERROR_DEVICE_NOT_FOUND = Device not found

DWORD SGFPM_CloseDevice(HSGFPM hFpm)

Closes the opened device. **SGFPM_OpenDevice()** must be called before this function is used.

- **Parameters**

- ***hFpm***

- The handle of the SGFPM object

- **Return values**

- SGFDX_ERROR_NONE = No error

DWORD SGFPM_GetDeviceInfo(HSGFPM hFpm, SGDeviceInfoParam* pInfo)

Gets device information from the driver (before device initialization)

- **Parameters**

- ***hFpm***

- The handle of the SGFPM object

- ***pInfo***

- A pointer to SGDeviceInfoParam. SGDeviceInfoParam is explained in [Chapter 5. Structure Reference](#).

- **Return values**

- SGFDX_ERROR_NONE = No error

DWORD SGFPM_Configure(HSGFPM hFpm, HWND hwnd)

Displays the driver's configuration dialog box

- **Parameters**

- ***hFpm***

- The handle of the SGFPM object

- ***hwnd***

- The parent window handle

- **Return values**

- SGFDX_ERROR_NONE = No error

DWORD SGFPM_SetBrightness(HSGFPM hFpm, DWORD brightness)

Controls brightness of image sensor

- **Parameters**

- ***hFpm***

- The handle of the SGFPM object

brightness

Brightness value (from 0 to 100)

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

DWORD SGFPM_SetLedOn(HSGFPM hFpm, bool on)

Turns optic unit LED on/off

- **Parameters**

hFpm

The handle of the SGFPM object

on

True: Turns on LED

False: Turns off LED

- **Return values**

SGFDX_ERROR_NONE = No error

DWORD SGFPM_GetImage(HSGFPM hFpm, BYTE* buffer)

Captures a 256 gray-level fingerprint image from the reader. The image size can be retrieved by calling **SGFPM_GetDeviceInfo()**. **SGFPM_GetImage()** does not check for image quality. To get image quality of a captured image, use **SGFPM_GetImageQuality()**. To get the approximate image quality while capturing, use **GetImageEx()**.

- **Parameters**

hFpm

The handle of the SGFPM object

buffer

A pointer to the buffer containing a fingerprint image. The image size can be retrieved by calling **GetDeviceInfo()**.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_WRONG_IMAGE = Capture image is not a real fingerprint image

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_LINE_DROPPED = Image data lost

DWORD SGFPM_BeginGetImage(HSGFPM hFpm)

Prepares for **SGFPM_GetImage()** to start capture. Must call **SGFPM_EndGetImage()** later.

- **Parameters**

hFpm

The handle of the SGFPM object

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_UNSUPPORTED_DEV = Not supported

DWORD SGFPM_EndGetImage(HSGFPM hFpm)

Closes **SGFPM_GetImage()** to finish capture

- **Parameters**

hFpm

The handle of the SGFPM object

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 SGFDX_ERROR_UNSUPPORTED_DEV = Not supported

DWORD SGFPM_GetImageQuality(HSGFPM hFpm, DWORD width, DWORD height, BYTE* imgBuf, DWORD* quality)

Gets the quality of a captured (scanned) image. The value is determined by two factors. One is the ratio of the fingerprint image area to the whole scanned area, and the other is the ridge quality of the fingerprint image area. A quality value of 50 or higher is recommended for registration. A quality value of 40 or higher is recommended for verification.

Note: The returned quality value is different from the value used in **SGFPM_GetImageEx()**. The quality value in **SGFPM_GetImageEx()** represents only the ratio of the fingerprint image area to the whole scanned area. **SGFPM_GetLastImageQuality()** can be called instead, which will have the same quality value as **SGFPM_GetImageQuality**.

- **Parameters**
 - hFpm**
The handle of the SGFPM object
 - width**
Image width in pixels
 - height**
Image height in pixels
 - imgBuf**
Fingerprint image data
 - quality**
The return value indicating image quality
- **Return values**
 - SGFDX_ERROR_NONE = No error
 - SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

DWORD SGFPM_GetLastImageQuality(HSGFPM hFpm, DWORD* quality)

Gets the quality of the last captured (scanned) image. The value is determined only by one fact, which is the ridge quality of the fingerprint image area. A quality value of 70 or higher is recommended for registration. A quality value of 50 or higher is recommended for verification. The value ranges from zero to 100.

Note: Not all devices support this function. Therefore, the return value should be checked.

- **Parameters**
 - hFpm**
The handle of the SGFPM object
 - quality**
The return value indicating image quality
- **Return values**
 - SGFDX_ERROR_NONE = No error
 - SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 - SGFDX_ERROR_UNSUPPORTED_DEV = Not supported

DWORD SGFPM_GetImageEx(HSGFPM hFpm, BYTE* buffer, DWORD time = 0, HWND dispWnd, DWORD quality)

Captures fingerprint images from the reader until the quality of the image is greater than the value of the quality parameter. The captured fingerprint is a 256 gray-level image. Image size can be retrieved by calling

the **SGFPM_GetDeviceInfo()** function. A quality value of 50 or higher is recommended for registration. A quality value of 40 or higher is recommended for verification.

Note: The returned quality value is different from the value used in **SGFPM_GetImage()**. The quality value in **GetImageEx()** represents only the ratio of the fingerprint image area to the whole scanned area.

- **Parameters**

- **hFpm**

- The handle of the SGFPM object

- **buffer**

- Pointer to buffer containing a fingerprint image

- **timeout**

- The timeout value (in milliseconds) used to specify the amount of time the function will wait for a valid fingerprint to be input on the fingerprint reader

- **dispWnd**

- Window handle used for displaying fingerprint images

- **quality**

- The minimum quality value of an image, used to determine whether to accept the captured image

- **Return values**

- SGFDX_ERROR_NONE = No error

- SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

- SGFDX_ERROR_LINE_DROPPED = Image data lost

- SGFDX_ERROR_TIME_OUT = No valid fingerprint captured in the given time

DWORD SGFPM_EnableAutoOnEvent (HSGFPM hFpm, BOOL enable, HWND hwnd, void* reserved)

Allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. **SGFPM_EnableAutoOnEvent()** enables or disables the Auto-On function. Once Auto-On is enabled, the application can receive a message from the device driver whenever an Auto-On event occurs in the reader. (Not supported by FDU02-based readers.)

When calling **SGFPM_EnableAutoOnEvent()**, pass the handle of the window that will receive the **Auto-On** message. The **Auto-On** message is defined as 0x8100 in sgfplib.h.

- **Parameters**

- **hFpm**

- The handle of the SGFPM object

- **enable**

- TRUE: Auto-On is enabled

- FALSE: Auto-On is disabled

- **hwnd**

- Window handle to receive Auto-On message

- **reserved**

- Not used

- **Return values**

- SGFDX_ERROR_NONE = No error

- SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

- **Remarks**

- When the application receives an Auto-On message, wParam will have event type (Finger ON or OFF) and lParam will have information of the device from which the event occurred.

- **wParam:**

- Contains event type.

- SGDEVEVNET_FINGER_ON(1) = Finger is on the sensor

SGDEVEVNET_FINGER_OFF(0) = Finger is removed from the sensor

IParam:

Contains device information. The device information is contained in SGDeviceInfoParam.

4.4. Extraction Functions

DWORD SGFPM_GetMaxTemplateSize(HSGFPM hFpm, DWORD* size)

Gets the maximum size of a fingerprint template (view or sample). Use this function before using **SGFPM_CreateTemplate()** to obtain an appropriate buffer size. If the template format is SG400, it returns a fixed length size of 400.

Note: The returned template size means the maximum size of one view or sample.

- **Parameters**

hFpm

The handle of the SGFPM object

size

The pointer to contain template size

- **Return values**

SGFDX_ERROR_NONE = No error

DWORD SGFPM_CreateTemplate(HSGFPM hFpm, FingerInfo* fpInfo, BYTE *rawImage, BYTE* minTemplate)

Extracts minutiae from a fingerprint image to form a template having the default format

- **Parameters**

hFpm

The handle of the SGFPM object

fpInfo

Fingerprint information stored in a template. For **ANSI378** templates, this information can be retrieved from the template using **GetAnsiTemplateInfo()**. For **ISO19794-2** templates, this information can be retrieved from the template using **GetIsoTemplateInfo()**. For **ISO19794-2 Compact** templates, this information can be retrieved from the template using **GetIsoCompactTemplateInfo()**. For **SG400** templates, this information cannot be seen in the template. For more information about the structure, refer to [Section 5.4 SGFingerInfo](#).

rawImg

256 Gray-level fingerprint image data

minTemplate

Pointer to buffer containing minutiae data extracted from a fingerprint image

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FEAT_NUMBER = Inadequate number of minutia

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = 103 = Error while decoding template 1

SGFDX_ERROR_INVALID_TEMPLATE2 = 104 = Error while decoding template 2

DWORD SGFPM_CreateTemplateDev(HSGFPM hFpm, DWORD* size)

Captures a fingerprint image and Extracts minutiae to form a template having the default format.

To get the template, call **SGFPM_GetTemplateDev()** with a buffer having the size. (U20-ASF-BT only)

- **Parameters**

hFpm

The handle of the SGFPM object

size

A pointer to the size of the template created.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FUNCTION_FAILED

DWORD SGFPM_GetTemplateDev(HSGFPM hFpm, BYTE* minTemplate)

Gets the template. (U20-ASF-BT only)

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate

A pointer to the buffer where the template created is copied

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FUNCTION_FAILED

DWORD SGFPM_GetTemplateSize(HSGFPM hFpm, BYTE* minTemplate, DWORD* size)

Gets template size. If the template format is SG400, it will return 400. If the template format is ANSI378, ISO19794-2, or ISO19794-2 Compact, template size may vary.

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate

Pointer to buffer containing minutiae data extracted from a fingerprint image

size

The pointer to contain template size

- **Return values**

SGFDX_ERROR_NONE = No error

4.5. Matching Functions

DWORD SGFPM_MatchTemplate(HSGFPM hFpm, BYTE *minTemplate1, BYTE *minTemplate2, DWORD secuLevel, BOOL* matched)

Compares two sets of minutiae data of the same template format. The template format should be the same as that set by **SGFPM_SetTemplateFormat()** and should include only one sample. To match templates that have more than one sample, use **SGFPM_MatchTemplateEx()**, **SGFPM_MatchAnsiTemplate()**, **SGFPM_MatchIsoTemplate()**, or **SGFPM_MatchIsoCompactTemplate()**. It returns TRUE or FALSE as a matching result (**matched**). The security level (**secuLevel**) affects matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate1

A pointer to the buffer containing minutiae data extracted from a fingerprint image

minTemplate2

A pointer to the buffer containing minutiae data extracted from a fingerprint image

secuLevel

A security level as specified in “fplibnew.h” by one the following nine security levels: SL_LOWEST, SL_LOWER, SL_LOW, SL_BELOW_NORMAL, SL_NORMAL (recommended), SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST.

matched

Contains matching result. If passed templates are the same, **TRUE** is returned. If not, **FALSE** is returned.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

DWORD SGFPM_MatchTemplateEx(HSGFPM hFpm, BYTE* minTemplate1, WORD templateType1, DWORD sampleNum1, BYTE* minTemplate2, WORD templateType2, DWORD sampleNum2, DWORD secuLevel, BOOL* matched)

Compares two sets of minutiae data, which can be of different template formats (SG400, ANSI378, ISO19794-2, or ISO19794-2 Compact). It returns TRUE or FALSE as a matching result (**matched**). The security level (**secuLevel**) affects matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate1

A pointer to the buffer containing minutiae data extracted from a fingerprint image

templateType1

Specifies format of minTemplate1 (TEMPLATE_FORMAT_SG400, TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT)

sampleNum1

Position of a sample to be matched in minTemplate1. If templateType1 is TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT, it can have a value from 0 to the number of

samples minus 1 in minTemplate1. If templateType1 is TEMPLATE_FORMAT_SG400, this value is ignored.

minTemplate2

A pointer to the buffer containing minutiae data extracted from a fingerprint image

templateType2

Specifies format of minTemplate2 (TEMPLATE_FORMAT_SG400, TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT)

sampleNum2

Position of a sample to be matched in minTemplate2. If templateType2 is TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT, it can have a value from 0 to the number of samples minus 1 in minTemplate2. If templateType2 is TEMPLATE_FORMAT_SG400, this value is ignored.

secuLevel

A security level as specified in "fplibnew.h" by one the following nine security levels: SL_LOWEST, SL_LOWER, SL_LOW, SL_BELOW_NORMAL, SL_NORMAL (recommended), SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER, and SL_HIGHEST.

matched

TRUE: Same template

FALSE: Not same template

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

DWORD SGFPM_GetMatchingScore(HSGFPM hFpm, BYTE* minTemplate1, BYTE* minTemplate2, DWORD* score)

Gets matching score of two sets of minutiae data of the same template format

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate1

A pointer to the buffer containing minutiae data extracted from a fingerprint image

minTemplate2

A pointer to the buffer containing minutiae data extracted from a fingerprint image

score

Matching score (from 0 to 199)

- **Returned values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

DWORD SGFPM_GetMatchingScoreEx(HSGFPM hFpm, BYTE* minTemplate1, WORD templateType1, DWORD sampleNum1, BYTE* minTemplate2, WORD templateType2, DWORD sampleNum2, DWORD* score);

Gets matching score of two sets of minutiae data, which can be of different template formats (SG400, ANSI378, ISO19794-2, or ISO19794-2 COMPACT)

- **Parameters**

hFpm

The handle of the SGFPM object

minTemplate1

A pointer to the buffer containing minutiae data extracted from a fingerprint image

templateType1

Specifies format of minTemplate1 (TEMPLATE_FORMAT_SG400, TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT)

sampleNum1

Position of a sample to be matched in minTemplate1. If templateType1 is TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT, it can have a value from 0 to the number of samples minus 1 in minTemplate1. If templateType1 is TEMPLATE_FORMAT_SG400, this value is ignored.

minTemplate2

A pointer to the buffer containing minutiae data extracted from a fingerprint image

templateType2

Specifies format of minTemplate2 (TEMPLATE_FORMAT_SG400, TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT)

sampleNum2

Position of a sample to be matched in minTemplate2. If templateType2 is TEMPLATE_FORMAT_ANSI378, TEMPLATE_FORMAT_ISO19794, or TEMPLATE_FORMAT_ISO19794_COMPACT, it can have a value from 0 to the number of samples minus 1 in minTemplate2. If templateType2 is TEMPLATE_FORMAT_SG400, this value is ignored.

score

Matching score (from 0 to 199)

- **Returned values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

4.6. Functions for ANSI378 Templates

DWORD SGFPM_GetTemplateSizeAfterMerge(HSGFPM hFpm, BYTE* ansiTemplate1, BYTE* ansiTemplate2, DWORD* size)

Calculates template size if two templates – ansiTemplate1 and ansiTemplate2 – are merged. Use this function to determine exact buffer size before using **SGFPM_MergeAnsiTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

ansiTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

ansiTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

size

Template size if two templates are merged

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in ansiTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in ansiTemplate2

DWORD SGFPM_MergeAnsiTemplate(HSGFPM hFpm, BYTE* ansiTemplate1, BYTE* ansiTemplate2, BYTE* outTemplate)

Merges two ANSI378 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of the two input templates (size of ansiTemplate1 + size of ansiTemplate2). Call **SGFPM_GetTemplateSizeAfterMerge()** to determine the exact buffer size for **outTemplate** before calling **SGFPM_MergeAnsiTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

ansiTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

ansiTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

outTemplate

The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **SGFPM_GetTemplateSizeAfterMerge()**.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in ansiTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in ansiTemplate2

DWORD SGFPM_MergeMultipleAnsiTemplate(HSGFPM hFpm, BYTE* inTemplates, DWORD nTemplates, BYTE* outTemplate)

Merges multiple ANSI378 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of all templates in **inTemplates**.

- **Parameters**

hFpm

The handle of the SGFPM object

inTemplates

A series of ANSI378 templates [ANSITemplate-1, ANSITemplate-2, ANSITemplate-3, ... ANSITemplate-n]

nTemplates

The number of templates in **inTemplates**

outTemplate

The buffer containing new merged template data. The buffer should be assigned by the application.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_GetAnsiTemplateInfo(HSGFPM hFpm, BYTE* ansiTemplate, SGANSITemplateInfo* templateInfo)

Gets information of an ANSI378 template. Call this function before **SGFPM_MatchAnsiTemplate()** to obtain information about a template.

- **Parameters**

hFpm

The handle of the SGFPM object

ansiTemplate

ANSI378 template

templateInfoThe buffer that contains template information. For more information see **SGANSITemplateInfo** structure.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_MatchAnsiTemplate(HSGFPM hFpm, BYTE* ansiTemplate1, DWORD sampleNum1, BYTE* ansiTemplate2, DWORD sampleNum2, DWORD secuLevel, BOOL* matched)

Compares two sets of ANSI378 templates. It returns TRUE or FALSE as a matching result (**matched**). The security level (**secuLevel**) affects matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

hFpm

The handle of the SGFPM object

ansiTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1Position of sample to be matched in **ansiTemplate1**. It can be from 0 to the number of samples minus 1 in **ansiTemplate1*****ansiTemplate2***

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2Position of sample to be matched in **ansiTemplate2**. It can be from 0 to the number of samples minus 1 in **ansiTemplate2*****secuLevel***

A security level as specified in "fplibnew.h" by one the following nine security levels: SL_LOWEST, SL_LOWER, SL_LOW, SL_BELOW_NORMAL, SL_NORMAL (recommended), SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST.

matched

TRUE: Same template

FALSE: Not same template

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in ansiTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in ansiTemplate2

DWORD SGFPM_GetAnsiMatchingScore(HSGFPM hFpm, BYTE* ansiTemplate1, DWORD sampleNum1, BYTE* ansiTemplate2, DWORD sampleNum2, DWORD* score)

Gets matching score

- **Parameters**

hFpm

The handle of the SGFPM object

ansiTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1Position of sample to be matched in **ansiTemplate1**. It can be from 0 to the number of samples minus 1 in **ansiTemplate1**

ansiTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2

Position of sample to be matched in ***ansiTemplate2***. It can be from 0 to the number of samples minus 1 in ***ansiTemplate2***

score

Matching score (from 0 to 199)

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in *ansiTemplate1*

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in *ansiTemplate2*

4.7. Functions for ISO19794-2 Templates

DWORD SGFPM_GetIsoTemplateSizeAfterMerge(HSGFPM hFpm, BYTE* isoTemplate1, BYTE* isoTemplate2, DWORD* size)

Calculates template size if two templates – *isoTemplate1* and *isoTemplate2* – are merged. Use this function to determine exact buffer size before using **SGFPM_MergelsoTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

size

Template size if two templates are merged

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in *isoTemplate1*

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in *isoTemplate2*

DWORD SGFPM_MergelsoTemplate(HSGFPM hFpm, BYTE* isoTemplate1, BYTE* isoTemplate2, BYTE* outTemplate)

Merges two ISO19794-2 templates and returns a new merged template. The size of the merged template (***outTemplate***) will be smaller than the sum of the sizes of the two input templates (size of *isoTemplate1* + size of *isoTemplate2*). Call **SGFPM_GetIsoTemplateSizeAfterMerge()** to determine the exact buffer size for ***outTemplate*** before calling **SGFPM_MergelsoTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

outTemplate

The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **SGFPM_GetIsoTemplateSizeAfterMerge()**.

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
 SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
 SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

DWORD SGFPM_MergeMultipleIsoTemplate(HSGFPM hFpm, BYTE* inTemplates, DWORD nTemplates, BYTE* outTemplate)

Merges multiple ISO19794-2 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of all templates in **inTemplates**.

- **Parameters**

hFpm

The handle of the SGFPM object

inTemplates

A series of ISO19794-2 templates [ISOTemplate-1, ISOTemplate-2, ISOTemplate-3, ... ISOTemplate-n]

nTemplates

The number of templates in **inTemplates**

outTemplate

The buffer containing new merged template data. The buffer should be assigned by the application.

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_GetIsoTemplateInfo(HSGFPM hFpm, BYTE* isoTemplate, SGISOTemplateInfo* templateInfo)

Gets information of an ISO19794-2 template. Call this function before **SGFPM_MatchIsoTemplate()** to obtain information about a template.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate

ISO19794-2 template

templateInfo

The buffer that contains template information. For more information see **SGISOTemplateInfo** structure.

- **Return values**

SGFDX_ERROR_NONE = No error
 SGFDX_ERROR_INVALID_PARAM = Invalid parameter used
 SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_MatchIsoTemplate(HSGFPM hFpm, BYTE* isoTemplate1, DWORD sampleNum1, BYTE* isoTemplate2, DWORD sampleNum2, DWORD secuLevel, BOOL* matched)

Compares two sets of ISO19794-2 templates. It returns TRUE or FALSE as a matching result (**matched**). The security level (**secuLevel**) affects matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1

Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2

Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**

secuLevel

A security level as specified in "fplibnew.h" by one the following nine security levels: SL_LOWEST, SL_LOWER, SL_LOW, SL_BELOW_NORMAL, SL_NORMAL (recommended), SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST.

matched

TRUE: Same template

FALSE: Not same template

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

DWORD SGFPM_GetIsoMatchingScore(HSGFPM hFpm, BYTE* isoTemplate1, DWORD sampleNum1, BYTE* isoTemplate2, DWORD sampleNum2, DWORD* score)

Gets matching score

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1

Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2

Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**

score

Matching score (from 0 to 199)

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

4.8. Functions for ISO19794-2 Compact Card Templates⁶

DWORD SGFPM_GetIsoCompactTemplateSizeAfterMerge(HSGFPM hFpm, BYTE* isoTemplate1, BYTE* isoTemplate2, DWORD* size)

Calculates template size if two templates – isoTemplate1 and isoTemplate2 – are merged. Use this function to determine exact buffer size before using **SGFPM_MergelsoCompactTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

size

Template size if two templates are merged

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

DWORD SGFPM_MergelsoCompactTemplate(HSGFPM hFpm, BYTE* isoTemplate1, BYTE* isoTemplate2, BYTE* outTemplate)

Merges two ISO19794-2 compact card templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of the two input templates (size of isoTemplate1 + size of isoTemplate2). Call **SGFPM_GetIsoCompactTemplateSizeAfterMerge()** to determine the exact buffer size for **outTemplate** before calling **SGFPM_MergelsoCompactTemplate()**.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

outTemplate

The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **SGFPM_GetIsoCompactTemplateSizeAfterMerge()**.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

DWORD SGFPM_MergeMultipleIsoCompactTemplate(HSGFPM hFpm, BYTE* inTemplates, DWORD nTemplates, BYTE* outTemplate)

Merges multiple ISO19794-2 compact card templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of all templates in **inTemplates**.

- **Parameters**

⁶ U20-ASF-BT (Bluetooth SPP and BLE) devices do not support this format.

hFpm

The handle of the SGFPM object

inTemplates

A series of ISO19794-2 compact card templates [ISOTemplate-1, ISOTemplate-2, ISOTemplate-3, ..., ISOTemplate-n]

nTemplates

The number of templates in ***inTemplates***

outTemplate

The buffer containing new merged template data. The buffer should be assigned by the application.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_GetIsoCompactTemplateInfo(HSGFPM hFpm, BYTE* isoTemplate, SGISOTemplateInfo* templateInfo)

Gets information of an ISO19794-2 compact card template. Call this function before **SGFPM_MatchIsoCompactTemplate()** to obtain information about a template.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate

ISO19794-2 compact card template

templateInfo

The buffer that contains template information. For more information, see **SGISOTemplateInfo** structure.

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

DWORD SGFPM_MatchIsoTemplate(HSGFPM hFpm, BYTE* isoTemplate1, DWORD sampleNum1, BYTE* isoTemplate2, DWORD sampleNum2, DWORD secuLevel, BOOL* matched)

Compares two sets of ISO19794-2 templates. It returns TRUE or FALSE as a matching result (**matched**). The security level (**secuLevel**) affects matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1

Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2

Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**

secuLevel

A security level as specified in "sgfplib.h" by one the following nine security levels: SL_LOWEST,

SL_LOWER, SL_LOW, SL_BELOW_NORMAL, SL_NORMAL (recommended),
SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST.

matched

TRUE: Same template

FALSE: Not same template

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

DWORD SGFPM_GetIsoMatchingScore(HSGFPM hFpm, BYTE* isoTemplate1, DWORD sampleNum1, BYTE*isoTemplate2, DWORD sampleNum2, DWORD* score)

Gets matching score

- **Parameters**

hFpm

The handle of the SGFPM object

isoTemplate1

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1

Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**

isoTemplate2

A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum2

Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**

score

Matching score (from 0 to 199)

- **Return values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

SGFDX_ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

4.9. Other

DWORD SGFPM_GetMinexVersion(HSGFPM hFpm, DWORD *extractor, DWORD* matcher)

Gets version of MINEX Compliant algorithms used in this SDK

- **Parameters**

hFpm

The handle of the SGFPM object

extractor

Version of MINEX Compliant extractor (template generator)

matcher

Version of MINEX Compliant matcher (template matcher)

- **Return values**

SGFDX_ERROR_NONE = No error

Chapter 5. Structure Reference

5.1. SGDeviceInfoParam

```
typedef struct tagSGDeviceInfoParam
{
    DWORD DeviceID;           // 0 - 9
    BYTE DeviceSN[SGDEV_SN_LEN+1]; // Device serial number, SN length = 15
    DWORD ComPort;           // USB readers => USB (0x3BC+1)
    DWORD ComSpeed;          // USB reader => 0
    DWORD ImageWidth;        // Image width
    DWORD ImageHeight;       // Image height
    DWORD Contrast;          // 0 ~ 100
    DWORD Brightness;        // 0 ~ 100
    DWORD Gain;              // Device dependent
    DWORD ImageDPI;          // Image resolution
    DWORD FWVersion;         // Firmware version
} SGDeviceInfoParam;
```

Description

Used when calling SGFPM_GetDeviceInfo()

Members

DeviceID	Device ID for USB readers only (0 - 9)
DeviceSN	Device serial number for USB readers. SGDEV_SN_LEN = 15
ComPort	Contains DeviceID for USB readers
ComSpeed	Not used
ImageWidth	Fingerprint image width in pixels
ImageHeight	Fingerprint image height in pixels
Brightness	Current brightness value (0-100)
Contrast	Current contrast value (0-100)
Gain	Amplification (1, 2, 4, or 8) of image brightness (a higher value yields darker images)
ImageDPI	Fingerprint image resolution in DPI
FWVersion	Device firmware version number for USB readers

5.2. SGDeviceInfo

```
#define SGDEV_NAME_LEN    15
#define SGDEV_ID_LEN      63
typedef struct tagSGDeviceInfo
{
    wchar_t Name[SGDEV_NAME_LEN + 1];    // null-terminated wide string
    wchar_t ID[SGDEV_ID_LEN + 1];        // null-terminated wide string
} SGDeviceInfo, *LP SGDeviceInfo;
```

Description

Used when calling SGFPM_FindDevices()

Members

Name	Device name for U20-ASF-BT (BLE) devices
ID	Device ID for U20-ASF-BT (BLE) devices

5.3. SGDeviceList

```
typedef struct tagSGDeviceList
{
    DWORD DevName;
    DWORD DevID;
    WORD DevType;
    BYTE DevSN[SGDEV_SN_LEN+1];
} SGDeviceList;
```

Description

Used to obtain the currently attached device list in SGFPM_EnumerateDevice()

Members

DevName	Device name (for example: SG_DEV_FDU05 for U20 or SG_DEV_FDU07 for U10)
DevID	Device ID for USB readers
DevType	Not used
DeviceSN	Device serial number for USB readers (SGDEV_SN_LEN = 15)

5.4. SGFingerInfo

```
typedef struct tagSGFingerInfo {
    WORD FingerNumber;
    WORD ViewNumber;
    WORD ImpressionType;
    WORD ImageQuality;
} SGFingerInfo;
```

Description

Used when calling **SGFPM_CreateTemplate()**. The provided information will be put into the template. For **ANSI378** or **ISO 19794-2** templates, this information can be seen from the template structure format. For **SG400** templates, this information cannot be seen in the template.

Members

FingerNumber

<u>Finger position number</u>	<u>Finger</u>
SG_FINGPOS_UK (0x00):	Unknown finger
SG_FINGPOS_RT (0x01):	Right thumb
SG_FINGPOS_RI (0x02):	Right index finger
SG_FINGPOS_RM (0x03):	Right middle finger
SG_FINGPOS_RR (0x04):	Right ring finger
SG_FINGPOS_RL (0x05):	Right little finger
SG_FINGPOS_LT (0x06):	Left thumb
SG_FINGPOS_LI (0x07):	Left index finger
SG_FINGPOS_LM (0x08):	Left middle finger
SG_FINGPOS_LR (0x09):	Left ring finger
SG_FINGPOS_LL (0x0A):	Left little finger

ViewNumber

Sample number for each finger (starts at 0)

ImpressionType

Impression type (should be 0 for SecuGen readers)

SG_IMPTYPE_LP (0x00):	Live-scan plain
SG_IMPTYPE_LR (0x01):	Live-scan rolled
SG_IMPTYPE_NP (0x02):	Non-live-scan plain
SG_IMPTYPE_NR (0x03):	Non-live-scan rolled

ImageQuality

Image quality value (0 – 100). To get an image quality, use **GetImageQuality()**.

5.5. SGANSITemplateInfo/SGISOTemplateInfo

```
typedef struct tagSGANSITemplateInfo {
    DWORD                TotalSamples;
    SGFingerInfo         SampleInfo[225];
} SGANSITemplateInfo, SGISOTemplateInfo;
```

Description

Used when calling **SGFPM_GetAnsiTemplateInfo()** or **SGFPM_GetIsoTemplateInfo()**. The provided information will be put into the template. For ANSI378 templates, this information can be seen from the template structure format. For SG400 templates, this information cannot be seen in the template. For ISO19794-2 templates, this information can be seen from the template structure format.

Members**TotalSamples**

Indicates the number of samples in a template. One template can have a maximum of 225 samples.
 Number of samples = Max finger number 15 * Max View Number 15 = 225

SampleInfo

Information of each sample in a template. Refer to **SGFingerInfo** structure.

Chapter 6. Constants

6.1. SGFDxDeviceName

Device Name	Value	Description
SG_DEV_FDU02	0x03	FDU02-based reader
SG_DEV_FDU03	0x04	FDU03 or SDU03-based reader
SG_DEV_FDU04	0x05	FDU04 or SDU04-based reader
SG_DEV_FDU05	0x06	U20-based reader
SG_DEV_FDU06	0x07	UPx-based reader
SG_DEV_FDU06AP	0x16	UPx-AP based reader
SG_DEV_FDU07	0x08	U10-based reader
SG_DEV_FDU08	0x0A	U20-A based reader
SG_DEV_FDU08A	0x11	U20-AP based reader
SG_DEV_FDU08AL	0x17	U20-AL based reader
SG_DEV_FDU09A	0x12	U30 based reader
SG_DEV_FDU10A	0x13	U-AIR contactless reader
SG_DEV_FDUSDA	0x0D	U20-ASF-BT (Bluetooth SPP) based reader
SG_DEV_FDUSDA_BLE	0x0E	U20-ASF-BT (Bluetooth BLE) based reader
SG_DEV_UNKNOWN	0x00	Not determined

6.2. SGPPPortAddr

Port Address	Value	Description
AUTO_DETECT	0	Auto detect
USB_AUTO_DETECT	0x3BC+1	USB Auto detect

6.3. SGFDxSecurityLevel

Security Level	Value	Description
SL_NONE	0	No Security
SL_LOWEST	1	Lowest
SL_LOWER	2	Lower
SL_LOW	3	Low
SL_BELOW_NORMAL	4	Below normal
SL_NORMAL	5	Normal
SL_ABOVE_NORMAL	6	Above normal
SL_HIGH	7	High
SL_HIGHER	8	Higher
SL_HIGHEST	9	Highest

6.4. SGFDxTemplateFormat

Template Format	Value	Description
TEMPLATE_FORMAT_ANSI378	0x0100	ANSI-INCITS 378-2004 format
TEMPLATE_FORMAT_SG400	0x0200	SecuGen proprietary format
TEMPLATE_FORMAT_ISO19794	0x0300	ISO/IEC 19794-2:2005 format
TEMPLATE_FORMAT_ISO19794_COMPACT	0x0400	ISO/IEC 19794-2:2005 compact card format

6.5. SGImpressionType

Security Level	Value	Description
SG_IMPTYPE_LP	0x00	Live-scan plain
SG_IMPTYPE_LR	0x01	Live-scan rolled
SG_IMPTYPE_NP	0x02	Non-live-scan plain
SG_IMPTYPE_NR	0x03	Non-live-scan rolled

6.6. SGFingerPosition

Security Level	Value	Description
SG_FINGPOS_UK	0x00	Unknown finger
SG_FINGPOS_RT	0x01	Right thumb
SG_FINGPOS_RI	0x02	Right index finger
SG_FINGPOS_RM	0x03	Right middle finger
SG_FINGPOS_RR	0x04	Right ring finger
SG_FINGPOS_RL	0x05	Right little finger
SG_FINGPOS_LT	0x06	Left thumb
SG_FINGPOS_LI	0x07	Left index finger
SG_FINGPOS_LM	0x08	Left middle finger
SG_FINGPOS_LR	0x09	Left ring finger
SG_FINGPOS_LL	0x0A	Left little finger

6.7. SGFDxErrorCode

Error Code	Value	Description
General Error Codes		
SGFDX_ERROR_NONE	0	No error
SGFDX_ERROR_CREATION_FAILED	1	SGFPM object creation failed
SGFDX_ERROR_FUNCTION_FAILED	2	Function call failed
SGFDX_ERROR_INVALID_PARAM	3	Invalid parameter used
SGFDX_ERROR_NOT_USED	4	Not used function
SGFDX_ERROR_DLLLOAD_FAILED	5	DLL loading failed
SGFDX_ERROR_DLLLOAD_FAILED_DRV	6	Device driver loading failed
SGFDX_ERROR_DLLLOAD_FAILED_ALGO	7	Algorithm DLL loading failed
SGFDX_ERROR_NO_LONGER_SUPPORTED	8	No longer supported
SGFDX_ERROR_DLLLOAD_FAILED_WSQ	9	Sgwsqlib.dll not found
Device Driver Error Codes		
SGFDX_ERROR_SYSLOAD_FAILED	51	Cannot find driver sys file
SGFDX_ERROR_INITIALIZE_FAILED	52	Chip initialization failed
SGFDX_ERROR_LINE_DROPPED	53	Image data lost
SGFDX_ERROR_TIME_OUT	54	GetImageEx() timeout
SGFDX_ERROR_DEVICE_NOT_FOUND	55	Device not found
SGFDX_ERROR_DRVLOAD_FAILED	56	Driver file load failed
SGFDX_ERROR_WRONG_IMAGE	57	Wrong image
SGFDX_ERROR_LACK_OF_BANDWIDTH	58	Lack of USB bandwidth
SGFDX_ERROR_DEV_ALREADY_OPEN	59	Device is already opened
SGFDX_ERROR_GETSN_FAILED	60	Serial number does not exist
SGFDX_ERROR_UNSUPPORTED_DEV	61	Unsupported device
SGFDX_ERROR_FAKE_FINGER*	62	Fake finger detected
SGFDX_ERROR_FAKE_INITIALIZE_FAILED*	63	Cannot initialize for fake detection

(*Fake detection functions currently only available for U20-based device.)

Extract & Matching Error Codes		
SGFDX_ERROR_FEAT_NUMBER	101	Inadequate number of minutiae
SGFDX_ERROR_INVALID_TEMPLATE_TYPE	102	Wrong template type
SGFDX_ERROR_INVALID_TEMPLATE1	103	Error in decoding template 1
SGFDX_ERROR_INVALID_TEMPLATE2	104	Error in decoding template 2
SGFDX_ERROR_EXTRACT_FAIL	105	Extraction failed
SGFDX_ERROR_MATCH_FAIL	106	Matching failed

License Error Codes		
SGFDX_ERROR_LICENSE_LOAD	501	Cannot find license file
SGFDX_ERROR_LICENSE_KEY	502	Invalid license key
SGFDX_ERROR_LICENSE_EXPIRED	503	Expired license key
SGFDX_ERROR_LICENSE_WRITE	504	Cannot write license (not used)

WSQ Error Codes		
SGFDX_ERROR_NO_IMAGE	600	Invalid image

6.8. Other Constants

- SGDEV_SN_LEN 15 // Device serial number length
- WM_APP_SGAUTOONEVENT 0x8100
- SGDEVEVNET_FINGER_OFF 0
- SGDEVEVNET_FINGER_ON 1
- SGDEV_NAME_LEN 15 // Device name length
- SGDEV_ID_LEN 63 // Device ID length

Appendix A. Using SGFPM Objects Directly

All SDK functions are integrated into the SGFPM class. To access the SGFPM class (not by handle), get the pointer of an SGFPM object using **CreateSGFPMObject()**. When you have finished using the SGFPM object, destroy the SGFPM object using **DestroySGFPMObject()**.

A.1. Creating an SGFPM object

To get a pointer to an SGFPM object, use **CreateSGFPMObject()**. To create and use the SGFPM object, it should be called.

```
SGFPM *g_Fpm;                                // SGFPM object pointer
DWORD err = CreateSGFPMObject(&g_Fpm);
if (err != SGFDX_ERROR_NONE)
    g_Fpm->Init(SG_DEV_FDU05);
```

A.2. Destroying an SGFPM object

When exiting a program, you must destroy the SGFPM object with **DestroySGFPMObject()**.

```
DestroySGFPMObject(g_Fpm);                    // Destroys SGFPM object
```

A.3. Accessing other member functions

All member functions are nearly the same as the functions in the C style APIs described in [Chapter 3](#). The only difference is that functions are accessed through object pointers, not handles.

```
g_Fpm->Init(devName);
g_Fpm->InitEx2(width, height, dpi, path_to_license_file);
g_Fpm->SetTemplateFormat(format) // Default format is SG400
g_Fpm->EnumerateDevice(ndevs, devList)
g_Fpm->OpenDevice(devId)
g_Fpm->CloseDevice()
g_Fpm->GetDeviceInfo(pDeviceInfo)
g_Fpm->Configure(hwnd)
g_Fpm->SetBrightness(brightness)
g_Fpm->SetLedOn(onoff)
g_Fpm->GetImage(buffer)
g_Fpm->GetImageEx(buffer, time = 0, dispWnd, quality)
g_Fpm->GetImageQuality(width, height, imgBuf, quality)
g_Fpm->EnableAutoOnEvent(enable, hwnd, reserved)
g_Fpm->GetMaxTemplateSize(size)
g_Fpm->CreateTemplate(fpInfo, rawImage, minTemplate)
g_Fpm->GetTemplateSize(buf, size)
g_Fpm->MatchTemplate(minTemplate1, minTemplate2, secuLevel, matched)
```

```

g_Fpm->GetMatchingScore(min1, min2, score)
g_Fpm->GetTemplateSizeAfterMerge(ansiTemplate1, ansiTemplate2, size)
g_Fpm->MergeAnsiTemplate(ansiTemplate1, ansiTemplate2, outTemplate)
g_Fpm->MergeMultipleAnsiTemplate(inTemplates, nTemplates, outTemplate)
g_Fpm->GetAnsiTemplateInfo(BYTE* ansiTemplate, SGANSITemplateInfo* templateInfo)
g_Fpm->MatchAnsiTemplate(ansiTemplate1, sampleNum1,
                        ansiTemplate2, sampleNum2, secuLevel, matched)
g_Fpm->GetAnsiMatchingScore(ansiTemplate1, sampleNum1, ansiTemplate2, sampleNum2,
                           score)
g_Fpm->MatchTemplateEx(minTemplate1, templateType1, sampleNum1, minTemplate2,
                      templateType2, sampleNum2, secuLevel,
                      matched)
g_Fpm->GetMatchingScoreEx(minTemplate1, templateType1, sampleNum1, minTemplate2,
                        templateType2, sampleNum2, score)
g_Fpm->GetMinexVersion(extractor, matcher)

// ISO19794-2
g_Fpm->MergeIsoTemplate(isoTemplate1, isoTemplate2, outTemplate)
g_Fpm->MergeMultipleIsoTemplate(inTemplates, nTemplates, outTemplate)
g_Fpm->GetIsoTemplateInfo(isoTemplate, templateInfo)
g_Fpm->MatchIsoTemplate(isoTemplate1, sampleNum1,
                      isoTemplate2, sampleNum2,
                      secuLevel, matched)
g_Fpm->GetIsoMatchingScore(isoTemplate1, sampleNum1,
                          isoTemplate2, sampleNum2, score)

// ISO19794-2 Compact Card Template
g_Fpm->MergeIsoCompactTemplate(isoTemplate1, isoTemplate2, outTemplate)
g_Fpm->MergeMultipleIsoCompactTemplate(inTemplates, nTemplates, outTemplate)
g_Fpm->GetIsoCompactTemplateInfo(isoTemplate, templateInfo)
g_Fpm->MatchIsoCompactTemplate(isoTemplate1, sampleNum1,
                              isoTemplate2, sampleNum2,
                              secuLevel, matched)
g_Fpm->GetIsoCompactMatchingScore(isoTemplate1, sampleNum1,
                                isoTemplate2, sampleNum2, score)

// U20-ASF-BT (Bluetooth SPP and BLE)
g_Fpm->CreateTemplateDev(outSize);
g_Fpm->GetTemplateDev(outTemplate);
g_Fpm->GetTemplateFormatDev(outFormat);
g_Fpm->SetTemplateFormatDev(inFormat);

// U20-ASF-BT (Bluetooth BLE) only
g_Fpm->FindDevices(&ndevs, &devList, timeout);
g_Fpm->CancelFind;
g_Fpm->OpenDevice2(devId);

```

Appendix B. Using .NET Library

For information about usage and programming using SecuGen's .NET library, please refer to the separate document: ***FDx SDK Pro .NET Programming Manual***.

Appendix C. Getting a Template from Bluetooth Devices (U20-ASF-BT)

Due to the data size of a fingerprint image, it can take longer to transfer a fingerprint image wirelessly from U20-ASF-BT Bluetooth devices than it would to transfer from USB devices connected to the host. Therefore, for faster performance, it is recommended to transfer fingerprint templates, which have much smaller data size, instead of the images. Among several built-in capabilities, U20-ASF-BT Bluetooth devices can perform image capture, image processing, and template generation within the device.

```

/*
 * template formats for SDA
 */
#define TEMPLATE_ANSI378                0x0100
#define TEMPLATE_SG400                  0x0200 // default
#define TEMPLATE_ISO19794_2            0x0300

void get_template(HSGFPM hFPM, WORD format) {
    DWORD dwErr = SGFDX_ERROR_NONE;

    // get the current template format
    WORD current_template_format = 0;
    dwErr = SGFPM_GetTemplateFormatDev(hFPM, &current_template_format);
    assert(dwErr == SGFDX_ERROR_NONE);

    // set template
    WORD template_format = format;
    dwErr = SGFPM_SetTemplateFormatDev(hFPM, template_format);
    assert(dwErr == SGFDX_ERROR_NONE);
    std::cout << "current template format=0x" << std::hex << current_template_for
mat
        << ", new template format=0x" << template_format << std::endl;

    // capture and create a template
    DWORD template_size = 0;
    dwErr = SGFPM_CreateTemplateDev(hFPM, &template_size);
    assert(dwErr == SGFDX_ERROR_NONE);
    assert(template_size > 0);

    // get template data
    if (template_size > 0) {
        char *template_data = new char[template_size];
    }
}

```

```

        dwErr = SGFPM_GetTemplateDev(hFPM, (BYTE*)template_data);
        assert(dwErr == SGFDX_ERROR_NONE);

        save_template(template_data, template_size, template_format);
        delete[] template_data;
    }
}

/*
 * main
 */
int main()
{
    DWORD rc = SGFDX_ERROR_NONE;
    HSGFPM hFPM = NULL;

    // Create a SGFPM object
    rc = SGFPM_Create(&hFPM);
    if (rc != SGFDX_ERROR_NONE) {
        std::cout << "ERR: SGFPM_Create returns " << rc << std::endl;
    } else {
        // Initialize the SGFPM Object
        rc = SGFPM_Init(hFPM, SG_DEV_FDUSDA); // or SG_DEV_FDUSDA_BLE

        if (rc != SGFDX_ERROR_NONE) {
            std::cout << "ERR: SGFPM_Init returns " << rc << std::endl;
        } else {
            Int comport = 6;
            rc = SGFPM_OpenDevice(hFPM, comport);

            // U20-ASF-BT BLE device : BT-3175
            /*
            std::wstring id = { L"BluetoothLE#BluetoothLEac:d1:b8:d0:d6:e4-
cc:35:5a:ff:f0:37" };
            rc = SGFPM_OpenDevice2(hFPM, (wchar_t*)id.c_str());
            */

            if (rc != SGFDX_ERROR_NONE) {
                std::cout << "ERR: SGFPM_OpenDevice returns " << rc << std::endl;
            } else {
                get_template(hFPM, TEMPLATE_SG400);

                // Close the device
                SGFPM_CloseDevice(hFPM);
            }
        }
    }
}

```

```
    }  
}  
// Destory the SGFPM object  
SGFPM_Terminate(hFPM);  
}
```