



Programming Manual for FDx SDK Pro for UNIX/Linux

For applications using SecuGen® fingerprint modules

SG1-0034A-005 (04/18)

Copyright © 1998-2018 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, FDU03, FDU04, SDU03, SDU04, Smart Capture U10, U20, and UPx 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. Installation.....	5
2.1. Installation	5
2.2. 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	8
3.5. Getting Device Information	9
3.6. Capturing a Fingerprint Image	9
3.7. Getting Image Quality	10
3.8. Controlling Brightness.....	11
3.9. Creating a Template.....	11
3.10. Matching Templates	12
3.11. Registration process.....	16
3.12. Verification Process.....	17
3.13. Getting Matching Score	18
3.14. Using Auto-On™	19
3.15. Template Format	21
3.16. Manipulating ANSI378 Templates	23
3.17. Manipulating ISO19794-2 Templates	26
3.18. Getting Version Information of MINEX Compliant Algorithms.....	29
Chapter 4. Function Reference	30
4.1. SGFPM Creation and Termination	30
4.2. Initialization.....	31
4.3. Device and Capturing Functions	33

4.4. Extraction Functions	39
4.5. Matching Functions.....	40
4.6. Functions for ANSI378 Templates.....	44
4.7. Functions for ISO19794-2 Templates.....	48
4.8. Other	52
Chapter 5. Structure Reference	53
5.1. SGDeviceInfoParam	53
5.2. SGDeviceList.....	54
5.3. SGFingerInfo.....	54
5.4. SGANSITemplateInfo/SGISOTemplateInfo.....	55
Chapter 6. Constants.....	56
6.1. SGFDxDeviceName.....	56
6.2. SGPPPortAddr	56
6.3. SGFDxSecurityLevel.....	56
6.4. SGFDxTemplateFormat	57
6.5. SGImpressionType	57
6.6. SGFingerPosition	57
6.7. SGFDxErrorCode.....	58
6.8. Other Constants	58
Appendix A. Using SGFPM Objects Directly	59
A.1. Creating an SGFPM object	59
A.2. Destroying an SGFPM object	59
A.3. Accessing other member functions	59

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 to program and gives the most development flexibility among all SecuGen SDKs.

1.1. Features

- Uses SecuGen's new and improved next-generation algorithms
- Supports three kinds of fingerprint minutiae formats (or templates):
 - SG400: SecuGen's proprietary fingerprint minutiae format
 - ANSI378: Finger Minutiae Format for Data Exchange (ANSI-INCITS 378-2004)
 - ISO19794-2: Biometric Data Interchange Formats--Finger Minutiae Data (ISO/IEC 19794-2:2005)
- Provides low-level APIs for image capture, feature extraction and matching
- The following extraction and matching algorithms, which are incorporated in libsgfpamx.so 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)
- Gives a high degree of flexibility to developers of all kinds of applications and is easy to use

1.2. System Requirements

The SecuGen fingerprint reader captures a fingerprint image that is digitized 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. Supported SecuGen USB fingerprint readers and sensors:

- FDU03 sensor class: Hamster Plus and FDU03FR, FDU03FRS, SDU03M, or SDU03P sensors
- FDU04 sensor class: Hamster IV and FDU04 or SDU04P sensors
- FDU05 sensor class: Hamster Pro 20 and U20 sensors
- FDU06-sensor class: Hamster Pro and UPx sensors
- FDU07-sensor class: Hamster Pro 10 and U10 sensors

System requirements:

- IBM-compatible PC 486 or later
- 1 USB port (USB 2.0 required for FDU04, U20 and UPx-class readers)
- 64 MB RAM
- 80 MB available hard disk space
- Linux Kernel 2.6

Chapter 2. Installation

2.1. Installation

1. Unzip the FDx SDK Pro for UNIX/LINUX on your system.
2. Change directory to <installdir>/lib/target and run “make uninstall install”

2.2. Included Files

After unzipping the FDx SDK Pro for UNIX/LINUX, the following files are copied to your extraction directory (installdir).

<installdir>/readme.txt	Latest information about current release
<installdir>/lib/target	Runtime libraries
libsgfdu03.so	FDU03 device driver
libsgfdu04.so	FDU04 device driver
libsgfdu05.so	FDU05 device driver
libsgfdu06.so	FDU06 device driver
libsgfdu07.so	FDU07 device driver
libsgfpamx.so	Fingerprint algorithm module for extraction & matching (MINEX Compliant)
libsgfplib.so	Main module
Makefile	Makefile to install library files

<installdir>/bin/target directory

Demo applications linked with either FDU03, FDU04, FDU05 or FDU06 shared libraries

auto_on_fdu03	Demo console application for auto on using:	Hamster Plus
auto_on_fdu04		Hamster IV
auto_on_fdu05		Hamster Pro 20
auto_on_fdu06		Hamster Pro
auto_on_fdu07		Hamster Pro 10
fpmatcher_fdu03	GTK+ Demo program for fingerprint capture, extract and match using:	Hamster Plus
fpmatcher_fdu04		Hamster IV
fpmatcher_fdu05		Hamster Pro 20
fpmatcher_fdu06		Hamster Pro
fpmatcher_fdu07		Hamster Pro 10
multidev_fdu03	Demo console application for multiple device test using:	Hamster Plus
multidev_fdu04		Hamster IV
multidev_fdu05		Hamster Pro 20
multidev_fdu06		Hamster Pro
multidev_fdu07		Hamster Pro 10
sgd2_fdu03		Hamster Plus

sgd2_fdu04	GTK+ Demo program for fingerprint capture using:	Hamster IV
sgd2_fdu05		Hamster Pro 20
sgd2_fdu06		Hamster Pro
sgd2_fdu07		Hamster Pro 10
sgfplibtest_fdu03	Demo console application for fingerprint capture, extract and match using:	Hamster Plus
sgfplibtest_fdu04		Hamster IV
sgfplibtest_fdu05		Hamster Pro 20
sgfplibtest_fdu06		Hamster Pro
sgfplibtest_fdu07		Hamster Pro 10
test_auto_on_fdu03	Driver console application for auto on fingerprint capture using:	Hamster Plus
test_auto_on_fdu04		Hamster IV
test_auto_on_fdu05		Hamster Pro 20
test_auto_on_fdu06		Hamster Pro
test_auto_on_fdu07		Hamster Pro 10
test_fdu03	Driver console application for fingerprint capture using:	Hamster Plus
test_fdu04		Hamster IV
test_fdu05		Hamster Pro 20
test_fdu06		Hamster Pro
test_fdu07		Hamster Pro 10

<installdir>/include directory SDK library header file

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

<installdir>/sgfplibtest Sample source code for sgfplibtest

<installdir>/sgd2 Sample source code for sgd2

<installdir>/fpmatcher Sample source code for fpmatcher

/usr/local/lib

Runtime modules installed by running “make uninstall install” in <installdir>/lib/target

libsgfdu03.so	FDU03 device driver (Hamster Plus)
libsgfdu04.so	FDU04 device driver (Hamster IV)
libsgfdu05.so	FDU05 device driver (Hamster Pro 20)
libsgfdu06.so	FDU06 device driver (Hamster Pro)
libsgfdu07.so	FDU07 device driver (Hamster Pro 10)
libsgfpamx.so	MINEX compliant fingerprint algorithm module for extraction & matching
libsgfplib.so	Main module

Chapter 3. Programming in C/C++

SecuGen's FDX SDK *Pro* was designed for ease in programming and 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_InitEx()**. **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_InitEx()** takes image width, image height and resolution as parameters. Call **SGFPM_InitEx()** 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)
SGDEV_FDU03	4	FDU03 / SDU03 USB driver	260*300
SGDEV_FDU04	5	FDU04 / SDU04 USB driver	258*336
SGDEV_FDU05	6	U20 USB driver	300*400
SGDEV_FDU06	7	UPx USB driver	260*300

SGDEV_FDU07	8	U10 USB driver	252*330
-------------	---	----------------	---------

- **SGFPM_Init()**

```
DWORD devname = SG_DEV_FDU03;
err = SGFPM_Init(m_hFPM, devname);
```

- **SGFPM_InitEx()**

```
DWORD image_width = 260;
DWORD image_height = 300;
DOWRD image_dpi = 500;
err = SGFPM_InitEx(m_hFPM, image_width, image_height, image_dpi);
```

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.

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

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


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_DevidSN = device_info.DeviceSN;
    m_ImgWidth = device_info.ImageWidth;
    m_ImgHeight = device_info.ImageHeight;
    m_Contrast = device_info.Contrast;
    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);
}
```

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](#).

- **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
}
```

3.7. Getting Image Quality

To determine the fingerprint image quality, use **GetImageQuality()**.

- **SGFPM_GetImageQuality()**

```
DWORD img_qlty;
SGFPM_GetImageQuality(hFPM, ImageWidth, m_ImageHeight, fp_image,
&img_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 or ISO19794-2 format may be merged. For more information about template formats, refer to [Section 3.15. Template Format](#). For more information about merging templates, refer to [Section 3.16. Manipulating ANSI378 Templates](#) and [Section 3.17. Manipulating ISO19794-2 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 = 1;

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, the *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.

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

```

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_VrfTemplatel;
err = SGFPM_GetMaxTemplateSize(m_hFPM, &m_MaxTemplateSize);
m_VrfTemplatel= 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_VrfTemplatel);

DWORD sl = SL_NORMAL; // Set security level depending on
applications.
DWORD err;
BOOL matched1, matched2;
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplatel, m_VrfTemplatel,
sl);
err = SGFPM_MatchTemplate(m_hFPM, m_RegTemplate2, m_VrfTemplatel,
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_RegTemplate1, m_RegTemplate1,
&score) == SGSGFDX_ERROR_NONE)
{
    if (score > 100)
        // Enroll these fingerprints to database
    else
        // Try again
}

```

To understand how matching scores correlate 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. Corresponding Matching Score

Constant	Value	Corresponding 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

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.

In the UNIX/Linux environment, message queues are used for communication between the device driver and the host application. When no finger is present the messages from the driver contain a zero. When a finger is present, the driver will send a non-zero value.

- **Enabling Auto-On**

```
[Example]
bool StartAutoOn(LPSGFPM m_sgflib)
{
    DWORD result;
    bool StartAutoOn = false;

    //////////////////////////////////////
    // Start Message Queue //////////////////////////////////////
    // Create unique key via call to ftok()
    //Create unique message key
    key = ftok(".", 'a'); //'a' is an arbitrary seed value

    // Open the queue - create if necessary
    if((msg_qid = msgget(key, IPC_CREAT|0660)) == -1)
        return false;

    //////////////////////////////////////
    // EnableAutoOnEvent(true)
    result = m_sgflib->EnableAutoOnEvent(true, &msg_qid, NULL);
    if (result != SGFDX_ERROR_NONE)
    {
        printf("FAIL - [%ld]\n", result);
    }
    else
    {
        StartAutoOn = true;
        printf("SUCCESS - [%ld]\n", result);
    }
    return StartAutoOn;
}
```

- **Disabling Auto-On**

```

bool StopAutoOn(LPSGFPM m_sgfpplib)
{
    DWORD result;
    bool StopAutoOn = false;

    //////////////////////////////////////
    // EnableAutoOnEvent(false)
    result = m_sgfpplib->EnableAutoOnEvent(false, &msg_qid, NULL);
    if (result != SGFDX_ERROR_NONE)
    {
        printf("FAIL - [%ld]\n", result);
    }
    else
    {
        StopAutoOn = true;
        printf("SUCCESS - [%ld]\n", result);
    }

    //////////////////////////////////////
    // Remove Message Queue //////////////////////////////////////
    msgctl(msg_qid, IPC_RMID, 0);

    return StopAutoOn;
}

```

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

Check for finger presence in a separate thread or a while loop that pops messages from the message queue. The following function is an example:

```

long fingerPresent()
{
    int fingerPresent = 0;
    printf("Reading message queue ...\n");

#ifdef _FDU06
    qbuf.mtype = FDU06_MSG;
#endif
#ifdef _FDU05
    qbuf.mtype = FDU05_MSG;

```

```

#endif
#ifdef _FDU04
    qbuf.mtype = FDU04_MSG;
#endif
#ifdef _FDU03
    qbuf.mtype = FDU03_MSG;
#endif

    msgrcv(msg_qid, (struct msgbuf *)&qbuf, MAX_SEND_SIZE,
qbuf.mtype, 0);
    if (strlen(qbuf.mtext) > 0)
    {
        fingerPresent= atol(qbuf.mtext);
    }
    return fingerPresent;
}

```

3.15. Template Format

The FDx SDK *Pro* supports three types of fingerprint template formats:

- SecuGen’s proprietary template format (“SG400”)
- ANSI-INCITS 378-2004 “Finger Minutiae Format for Data Exchange” (“ANSI378”)
- ISO/IEC 19794-2:2005 “Biometric Data Interchange Formats-- Finger Minutiae Data” (“ISO19794-2”)

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

SG400 templates are encrypted for high security and have a size of 400 bytes. ANSI378 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 template, 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 under Subcommittee JTC 1 / SC 37 (Biometrics) http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38746.

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 with any template format:

- SGFPM_MatchTemplateEx()
- SGFPM_GetMatchingScoreEx()

- **Defining template format**

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

- **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);
```

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 merged template size will be less 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 merged template size will be less 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 merged template size will be less 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_MergelsoTemplate()**.

```

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 merged template size will be less 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. 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* phFPM)

Creates the SGFPM object internally

- **Parameters**

phFPM: 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**

pFPM: The handle of the SGFPM object

devName: Specifies the device name

SG_DEV_FDU03: device name for USB FDU03 and SDU03-based readers

SG_DEV_FDU04: device name for USB FDU04 and SDU04-based readers

SG_DEV_FDU05: device name for U20-based USB readers

SG_DEV_FDU06: device name for UPx-based USB readers

SG_DEV_FDU07: device name for U10-based USB 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_InitEx(HSGFPM hFpm, DWORD width, DWORD height, DWORD dpi)

Initializes SGFPM with image information. Use when running fingerprint algorithm module without a SecuGen reader.

- **Parameters**

pFPM: The handle of the SGFPM object

width: Image width in pixels

height: Image height in pixels

dpi: Image resolution in DPI

- **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 library

DWORD SGFPM_SetTemplateFormat(HSGFPM hFpm, WORD format)

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

- **Parameters**

pFPM: 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

- **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 (SG_DEV_FDU03 or SG_DEV_FDU04), it enumerates only the devices that belong to the specified device class.

- **Parameters**

pFPM: 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 5.2. SGDeviceList](#).

devName: Device name. Should be one of the following values:

SG_DEV_UNKNOWN = 0

SG_DEV_FDU03 = 0x04

SG_DEV_FDU04 = 0x05

SG_DEV_FDU05 = 0x06

SG_DEV_FDU06 = 0x07

- **Returned values**

SGFDX_ERROR_NONE = No error

SGFDX_ERROR_FUNCTION_FAILED = General function fail error

SGFDX_ERROR_INVALID_PARAM = Invalid parameter used

DWORD SGFPM_OpenDevice(HSGFPM hFpm, DWORD devId)

Initializes the fingerprint reader

- **Parameters**

pFPM: The handle of the SGFPM object

devId: Specifies the device ID for USB readers. The value can be from 0 to 9. The maximum number of supported readers attached at the same time is 10.

- **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**

pFPM: 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**

pFPM: 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**

pFPM: 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**

pFPM: The handle of the SGFPM object

brightness: Must be set to a 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**

pFPM: 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**

pFPM: 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_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.

- **Parameters**

pFPM: 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_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**

pFPM: 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.

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**

pFPM: The handle of the SGFPM object

enable: TRUE: enables Auto-On. FALSE: disables Auto-On

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

lParam: 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 fixed length size 400. **Note:** The returned template size means the maximum size of one view or sample.

- **Parameters**

pFPM: 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**

pFPM: The handle of the SGFPM object

fpInfo: Fingerprint information. It is stored in template. For **ANSI378** templates, this information can be retrieved from the template using **GetAnsiTemplateInfo()**. For **SG400** templates, this information cannot be seen in the template. For more information about the structure, refer to [section 5.3 SGFingerInfo](#). For **ISO19794** templates, this information can be retrieved from the template using **GetIsoTemplateInfo()**.

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_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 or ISO19794, template size may vary.

- **Parameters**

pFPM: 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()** or **SGFPM_MatchAnsiTemplate()**.

It returns TRUE or FALSE as a matching result(**matched**). Security level(**secuLevel**) affects matching result. The security level may be adjusted according to the security policy required by the user or organization

- **Parameters**

pFPM: 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, SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST. SL_NORMAL is recommended in usual case.

matched: Contains matching result. If passed templates are same templates, **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 or ANSI378). It returns TRUE or FALSE as a matching result (**matched**). Security level (**secuLevel**) affects matching result. The security level may be adjusted according to the security policy required by the user or organization.

- **Parameters**

pFPM: 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. Should be either TEMPLATE_FORMAT_SG400 or TEMPLATE_FORMAT_ANSI378.

sampleNum1: Position of a sample to be matched in minTemplate1. If templateType1 is TEMPLATE_FORMAT_ANSI378, it can have a value from 0 to (number of samples -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. Should be either TEMPLATE_FORMAT_SG400 or TEMPLATE_FORMAT_ANSI378.

sampleNum2: Position of a sample to be matched in minTemplate2. If templateType2 is TEMPLATE_FORMAT_ANSI378, it can have a value from 0 to (number of samples -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, SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER, and SL_HIGHEST. SL_NORMAL is recommended in usual case.

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**

pFPM: 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. Returned score has a value 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 tempateType1,
DWORD sampleNum1, BYTE* minTemplate2, WORD tempateType2, DWORD sampleNum2, DWORD*
score);
```

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

- Parameters

pFPM: 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. Should be either TEMPLATE_FORMAT_SG400 or TEMPLATE_FORMAT_ANSI378.

sampleNum1: Position of a sample to be matched in minTemplate1. If templateType1 is TEMPLATE_FORMAT_ANSI378, it can have a value from 0 to (number of samples -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. Should be either TEMPLATE_FORMAT_SG400 or TEMPLATE_FORMAT_ANSI378.

sampleNum2: Position of a sample to be matched in minTemplate2. If templateType2 is TEMPLATE_FORMAT_ANSI378, it can have a value from 0 to (number of samples -1) in minTemplate2. If templateType2 is TEMPLATE_FORMAT_SG400, this value is ignored.

score: Matching score. Returned score has a value 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**

pFPM: 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 minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

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

Merges two ANSI378 templates and returns a new merged template. The merged template (**outTemplate**) size will be less than 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**

pFPM: 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 minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

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

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

- **Parameters**

pFPM: 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

pFPM: The handle of the SGFPM object

ansiTemplate: ANSI378 template

templateInfo: The 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**). Security level (**secuLevel**) affects matching result. The security level may be adjusted according to the security policy required by the user or organization

- Parameters

pFPM: The handle of the SGFPM object

ansiTemplate1: A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1: Position of sample to be matched in **ansiTemplate1**. It can be from 0 to (number of samples -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 (number of samples -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, SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST. SL_NORMAL is recommended in usual case.

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**

pFPM: The handle of the SGFPM object

ansiTemplate1: A pointer to the buffer containing minutiae data. A template can have more than one sample.

sampleNum1: Position of sample to be matched in **ansiTemplate1**. It can be from 0 to (number of samples -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 (number of samples -1) in **ansiTemplate2**

score: Matching score. Returned score has a value 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_MergIsoTemplate()**.

- **Parameters**

pFPM: 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 minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

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

Merges two ISO19794-2 templates and returns a new merged template. The merged template (**outTemplate**) size will be less than 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_MergIsoTemplate()**.

- **Parameters**

pFPM: 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 minTemplate1

SGFDX_ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

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

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

- **Parameters**

pFPM: 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

pFPM: 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**). Security level (**secuLevel**) affects matching result. The security level may be adjusted according to the security policy required by the user or organization

- Parameters

pFPM: 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 (number of samples -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 (number of samples -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, SL_ABOVE_NORMAL, SL_HIGH, SL_HIGHER and SL_HIGHEST. SL_NORMAL is recommended in usual case.

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**

pFPM: 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 (number of samples -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 (number of samples -1) in ***isoTemplate2***

score: Matching score. Returned score has a value 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. Other

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

Gets version of MINEX Compliant algorithms used in this SDK

- **Parameters**

pFPM: 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;        // Parallel readers => PP address; USB readers =>
USB (0x3BC+1)
    DWORD ComSpeed;       // Parallel readers => PP mode; 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 (0 - 9)
- **DeviceSN:** Device serial number for USB readers. SGDEV_SN_LEN = 15
- **ComPort:** Not used
- **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 (higher values yields darker images)
- **ImageDPI:** Fingerprint image resolution in DPI
- **FWVersion:** Device firmware version number for USB readers

5.2. 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 (SG_DEV_FDU03 or SG_DEV_FDU04)
- **DevID:** Device ID for USB readers
- **DevType:** Not used
- **DeviceSN:** Device serial number for USB readers (SGDEV_SN_LEN = 15)

5.3. 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:** Fingerprint position number

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()`.

5.4. 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_UNKNOWN	0	Not determined
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_FDU07	0x08	U10-based reader

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

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	Library loading failed
SGFDX_ERROR_DLLLOAD_FAILED_DRV	6	Device driver loading failed
SGFDX_ERROR_DLLLOAD_FAILED_ALGO	7	Algorithm library loading failed
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
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

6.8. Other Constants

- SGDEV_SN_LEN 15 // Device serial number length
- WM_APP_SGAUTOONEVENT 0x8100
- SGDEVEVNET_FINGER_OFF 0
- SGDEVEVNET_FINGER_ON 1

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_FDU03);
```

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->InitEx(width, height, dpi);
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(minTemplatel, minTemplate2, secuLevel,
                    matched)
g_Fpm->GetMatchingScore(min1, min2, score)
g_Fpm->GetTemplateSizeAfterMerge(ansiTemplatel, ansiTemplate2,
                                size)
g_Fpm->MergeAnsiTemplate(ansiTemplatel, ansiTemplate2,
                        outTemplate)
g_Fpm->MergeMultipleAnsiTemplate(inTemplates, nTemplates,
                                outTemplate)
g_Fpm->GetAnsiTemplateInfo(BYTE* ansiTemplate,
                          SGANSITemplateInfo* templateInfo)
g_Fpm->MatchAnsiTemplate(ansiTemplatel, sampleNum1,
                        ansiTemplate2, sampleNum2, secuLevel,
                        matched)
g_Fpm->GetAnsiMatchingScore(ansiTemplatel, sampleNum1,
                           ansiTemplate2, sampleNum2, score)
g_Fpm->MatchTemplateEx(minTemplatel, tempateType1, sampleNum1,
                     minTemplate2, tempateType2, sampleNum2,
                     secuLevel, matched)
g_Fpm->GetMatchingScoreEx(minTemplatel, tempateType1, sampleNum1,
                         minTemplate2, tempateType2, sampleNum2,
                         score)
g_Fpm->GetMinexVersion(extractor, matcher)

// ISO19794-2

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

```