Programming Manual for FDx SDK ***Pro*** for Windows

For applications using SecuGen® fingerprint readers

SG1-0030A-013 (06/09)

Contents

Before You Begin

Biometrics Overview

Biometrics is an automated method of recognizing a person based on physical or behavioral characteristics. Biometric information that can be used to accurately identify people includes fingerprint, voice, face, iris, handwriting and hand geometry.

There are two key functions offered by a biometric system. One method is **identification**, a “one-to-many” matching process in which a biometric sample is compared sequentially to a set of stored samples to determine the closest match. The other is **verification**, a “one-to-one” matching process in which the biometric system checks previously enrolled data for a specific user to verify whether that individual is who he or she claims to be. The verification method provides the best combination of speed and security, especially where multiple users are concerned, and requires a user ID or other identifier for direct matching.

With an increasing reliance on online technology and other shared resources, the information age is quickly revolutionizing the way transactions are initiated and completed. Business transactions of all types are increasingly being handled online and remotely. This unprecedented growth in electronic transactions has underlined the need for a faster, more secure and more convenient method of user verification than passwords can provide.

Using biometric identifiers offers several advantages over traditional and current methods. This is because only biometric authentication is based on the identification of an intrinsic part of a human being. Tokens such as smart cards, magnetic stripe cards and physical keys, can be lost, stolen, duplicated or left behind. Passwords can be forgotten, shared, hacked or unintentionally observed by a third party. By eliminating all of these potential trouble spots, biometric technology can provide greater security, with convenience, needed for today’s complex electronic landscape.

Advantages of Using Fingerprints

The advantages of using fingerprints include widespread public acceptance, convenience and reliability. It takes little time and effort to scan one’s fingerprint with a fingerprint reader, and so fingerprint recognition is considered among the least intrusive of all biometric verification techniques. Ancient officials used thumbprints to seal documents thousands of years ago, and law enforcement agencies have been using fingerprint identification since the late 1800s. Fingerprints have been used so extensively and for so long, there is a great accumulation of scientific data supporting the idea that no two fingerprints are alike.

About SecuGen

SecuGen (www.secugen.com) provides biometric solutions for physical and network security employing advanced fingerprint recognition technology. The company’s comprehensive product line includes high quality optical fingerprint readers and sensor component, software and development kits that are used for a variety of innovative applications including Internet, enterprise network and desktop security, physical access control, time and attendance management and financial and medical records control. SecuGen patented products feature the industry’s longest warranty and are renowned for their accuracy, reliability and versatility. Based in Silicon Valley, SecuGen has been serving the global biometric community since 1998 and is an active member of the Biometrics Consortium (www.biometrics.org), the BioAPI Consortium (www.bioapi.org) and the International Biometric Industry Association (www.ibia.org).

About SecuGen Products

SecuGen Sensor Qualities

- **Excellent Image Quality:** Clear, distortion-free fingerprint images are generated using advanced, patent-pending optical methods. Quality imaging yields better sampling for minutiae data extraction.
- **Durability:** Mechanical strength tests show resistance to impact, shock and scratches.
- **Powerful Software:** Precise, fast processing algorithm ensures efficiency and reliability.
- **Ruggedness and Versatility:** Solid engineering and superior materials allows for use under extreme conditions.
- **Ergonomic Design:** Compact, modular design for seamless integration into small devices, ease of use and compatibility make it ideal for a broad range of applications.
- **Low Cost:** Products are developed to deliver high performance, zero maintenance at very affordable prices for general and industrial use.

Advantages of SecuGen Sensors Over Other Optical Sensors

- Unique optical method captures fine details, even from dry skin
- Extremely low image-distortion
- Reinforced materials
- Wear resistance
- Attractively small size
- Ease of integration
- Ready-to-use
- Low cost through longer life and no maintenance requirements

Advantages SecuGen Sensors Over Semiconductor (Capacitive) Sensors

- Non-metal, non-silicon components make it less susceptible to corrosion when exposed to salts, oil and moisture from skin and environment
- Superior surface properties eliminate need for costly coating and processing procedures
- Greater mechanical strength, wear-resistance and durability
- Broader range of applicability, especially for use in extreme conditions and climates
- Immunity from electrostatic discharge
- Low cost through longer life and no maintenance requirements

Strengths of SecuGen Software and Algorithms

- Unique image processing algorithm extracts fingerprint minutiae very accurately
- High signal-to-noise ratio processing algorithm screens out false features
- Highly efficient matching algorithm
- Fast overall process of extraction, matching and verification
- Encryption function to protect user privacy
- Compatibility with existing desktop, laptop PCs interface computers
- Ease in developing applications for various purposes

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 the SDKs in the Collection.

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 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)
- Gives a high degree of flexibility to developers of all kinds of applications and is easy to use

1.2. System Requirements

SecuGen USB 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. The following are the system requirements for SecuGen USB readers:

- IBM-compatible PC 486 or later
- 1 USB port (USB 2.0 required for FDU04-based readers)
- 64 MB RAM
- 80 MB available hard disk space
- Microsoft Windows Vista / XP / 2000 / 98 SE, Server 2003

Note: All SecuGen USB readers based on FDU02, FDU03, FDU04, and SDU03 sensors are supported in this SDK. FDU01-based readers are not supported.

Chapter 2. Installation

2.1. Installation

1. Insert the SecuGen SDK Collection CD into the CD-ROM drive of your computer. Open the file **setup.exe** located in the root directory *FDx SDK Pro Windows*.
2. Read the on-screen instructions and click ***Next*** to continue.
3. Click ***Yes*** to agree to the Software License Agreement. (If you do not agree, click ***No*** to cancel installation.)
4. Click ***Next*** to specify the default location, or ***Browse*** to select another folder. (C:*Program Files\SecuGen\FDx SDKPro is the default location.*)

2.2. Included Files

After installing the FDx SDK Pro for Windows, the following files are copied to your windows system directory and target directory.

Windows system directory
Runtime modules
- **sgfplib.dll** Main module
- **sgfpamx.dll** Fingerprint algorithm module for extraction & matching (MINEX Compliant)

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)
- **sgfplibx.ocx** ActiveX control

Bin\x64 directory
Runtime modules for 64-bit platform
- **sgfplib.dll** Main module
- **sgfpamx.dll** Fingerprint algorithm module for extraction & matching (MINEX Compliant)
- **sgfplibx.ocx** ActiveX control

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

Lib\x64 directory
SecuGen Fingerprint Module library for 64-bit platform
- **sgfplib.lib** Fingerprint Module import library

Samples directory

- HTML:
  - **Sample.htm**: Html sample code

- Visual Basic Sample Source Code
  - **Capture**: Sample code for handling devices, using ActiveX control
  - **Matching**: Sample code for matching templates, using ActiveX control

- Visual C++ Sample Source Code
  - **Device Test**: Sample code for image capture
  - **Matching**: Sample code for template matching
  - **MatchingANSI**: Sample code for matching ANSI378 templates and showing how they are handled
  - **MatchingISO**: Sample code for matching ISO19794-2 templates and showing how they are handled

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

- **SecuGen.FDxSDKPro.Windows.dll**
  - .NET assembly file wrapping sgfplib.dll
  - Note: The x64 .NET dll must be used for x64 platforms

DotNet\Samples directory

- Sample code using SecuGen.FDxSDKPro.Windows.dll
  - Matching sample written in C#
  - Matching sample showing ANSI378 template usage written 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 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_FDP02	1	Parallel device driver	260*300
SGDEV_FDU02	3	USB FDU02 driver	260*300
SGDEV_FDU03	4	USB FDU03 or SDU03 driver	260*300
SGDEV_FDU04	5	USB FDU04 driver	258*336

- **SGFPM_Init()**

```
DWORD  devname = SG_DEV_FDU02;
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. For example, if only one SecuGen USB fingerprint reader is connected to the PC, **devId** will be 0. If multiple SecuGen 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.

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

For most cases, if only one SecuGen 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 7. 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;
    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 5. 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, &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 or ISO19794-2 format may be merged. For more information about template formats, refer to Section 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 fucntion is the same as **SGFPM_MatchTemplateEx()** except that it supports only ISO19794-2 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_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_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 5.5. 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 the matching scores correlate with typical security levels, refer to the chart below. For more information about security levels, refer to Section 5.5. 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

Note: As of version 3.53 of FDx SDK *Pro*, the Corresponding Matching Scores have changed.

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 sgfplib.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 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 Interchage 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_MergeIsoTemplate()**
- **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_MergeIsoTemplate()**, 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_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 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. Programming with ActiveX Control

SecuGen's ActiveX control (sgfplibx.ocx) provides the user device facility and extraction and verification algorithms, helping programmers build applications using fingerprint-based identification & authentication applications easily and quickly in ActiveX supported languages including Visual Basic, Visual C++, Delphi and Borland C++ Builder. This chapter describes the use of ActiveX controls in this SDK. All SDK functions are integrated with sgfplibx.ocx.

sgfplibx.ocx is comprised of two controls, which you can use to access almost all functions in the SDK:

- **FpLibXCapture** Captures image data and extracts minutiae data from it
- **FpLibXVerify** Compares and verifies minutiae data with the stored minutiae data

4.1. Creating sgfplibx.ocx

In Visual Basic, you must add the ActiveX control by selecting "SGFpLibX ActiveX Control Module" from the "Components Pallet". FpLibXCapture and FpLibXVerify will then be added automatically. Other programming languages are similar. After registering the control, you can use the FpLibX control just like a button control.

4.2. Destroying sgfplibx.ocx

The sgfpLibx's controls are automatically deleted from memory when the program exits.

4.3. Opening Device

To initialize (open) the fingerprint reader, set your device type in the CodeName property field. Setting this value automatically initializes the fingerprint reader.

4.4. Image Capturing Operation

After successful initialization, fingerprint images can be captured from the reader using the **Capture()** method. The captured fingerprint is a 256 gray-level image. Image width and height can be retrieved by the **ImageWidth** and **ImageHeight** properties. Another method for acquiring images is with **GetImageData(),** in which case you must allocate memory for the image buffer and use the **ImageSize** property to retrieve the size of the image before using **GetImageData().** This method is rarely used, however, **GetMinutiaeData()** is the preferred method.

- Visual Basic

```
Dim buffer() as Byte
Redim buffer(FpLibXCapture.ImageSize) as Byte
If FpLibXCapture.Capture = True then
If FpLibXCapture.GetImageData(buffer) = True then
    ‘ To do your code
End If
End If
```

- C++

```
BYTE* buffer = new BYTE[FpLibXCapture.ImageSize];
VARIANT varBuffer;
```

```
varBuffer.vt = VT_BYREF|VT_UI1;
varBuffer.pbVal = buffer;
if (FpLibXCapture.Capture() == TRUE) {
   if (FpLibXCapture.GetImageData(varBuffer) == TRUE) {
       // To do your code
   }
}
delete[] buffer;
```

Similar to the **Capture()** method, the **LiveCapture()** method is also called to capture fingerprints from the reader. The more advanced **LiveCapture()** method captures fingerprint images continuously, checks the quality and ignores the image if no fingerprint is present or the quality is not acceptable. If an image of acceptable quality is captured within the given timeout parameter, **LiveCapture()** ends its processing.

- Visual Basic

```
Image_quality = 50
Timeout = 6000 ‘6 seconds
If FpLibXCapture.LiveCapture(Timeout, Image_quality) = True then
    ‘ To do your code
End If
```

- C++

```
short image_quality = 50;
long timeout = 6000;
if (FpLibXCapture.LiveCapture(timeout, image_quality) == TRUE) {
    // To do your code
}
```

4.5. Exposure Control

Set the **Brightness()** property to control the value of exposure.

```
FpLibXCapture.Brightness = 20 // Set from 0 to 100
                              // Default value is 40. It depends on the device.
```

Depending on the fingerprint reader and the specification of the host system, the brightness of the fingerprint images may vary. In this case, call **Configure()** method. Select “Auto Exposure” (or “Auto Tuning”) to configure the brightness automatically. This takes from 5 to 10 seconds to complete processing. The user’s finger must remain motionless on the fingerprint reader for the full duration of the process.

```
FpLibXCapture.Configure       // Calls driver built-in configuration dialog
```

4.6. Registration

Call the **GetMinutiaeData()** method to extract minutiae data from a fingerprint image. The extracted minutiae data can be saved to a file, database or buffer for further verification processing. Minutiae data are encrypted for high security.

- Visual Basic

```
Dim minData() as Byte
Redim minData(FpLibXCapture.MinutiaeSize) as Byte
If FpLibXCapture.LiveCapture(timeout, image_quality) = True then
   If FpLibXCapture.GetMinutiaeData(minData) = True then
        ‘ To do your code
   End If
```

```
End If
```

- C++

```
BYTE* minData = new BYTE[FpLibXCapture.MinutiaeSize];
VARIANT varBuffer;
varBuffer.vt = VT_BYREF|VT_UI1;
varBuffer.pbVal = minData;
if (FpLibXCapture.LiveCapture(timeout, image_quality) == TRUE) {
   if (FpLibXCapture.GetMinutiaeData(varBuffer) == TRUE) {
       // To do your code
   }
}
delete [] minData;
```

Two fingerprint images of the same finger must be captured, and the minutiae extracted from both before calling the **Register()** method in **FpLibXVerify** control to confirm the fingerprints. This confirming routine is analogous to a password confirmation routine required when entering a new password. This function has two parameters: two minutiae buffers. The two minutiae data packets are created by **GetImageData() and GetMinutiaeData(),** and the security level may be adjusted according to the security policy set by users.

- Visual Basic

```
Dim minData1() as Byte
Dim minData2() as Byte
Redim minData1(FpLibXCapture.MinutiaeSize) as Byte
Redim minData2(FpLibXCapture.MinutiaeSize) as Byte

If FpLibXCapture.LiveCapture(timeout, image_quality) = True then
   If FpLibXCapture.GetMinutiaeData(minData1) = True then
       ‘ To do your code
   End If
Else
   If FpLibXCapture.LiveCapture(timeout, image_quality) = True then
      If FpLibXCapture.GetMinutiaeData(minData2) = True then
          ‘ To do your code
     End If
   End If
   FpLibXVerify.SecurityLevel = 5
   If  FpLibXVerify.Register(minData1, minData2) = True then
         ‘ Save two minutia data to files.
End If
```

- C++

```
BYTE* minData1 = new BYTE[FpLibXCapture.MinutiaeSize];
BYTE* minData2 = new BYTE[FpLibXCapture.MinutiaeSize];
VARIANT varBuffer1, varBuffer2;
varBuffer1.vt = varBuffer2.vt = VT_BYREF|VT_UI1;
varBuffer1.pbVal = minData1;
varBuffer2.pbVal = minData2;
if (FpLibXCapture.LiveCapture(timeout, image_quality) == TRUE) {
   if (FpLibXCapture.GetMinutiaeData(varBuffer1) == TRUE) {
       // To do your code
   }
}
if (FpLibXCapture.LiveCapture(timeout, image_quality) == TRUE) {
   if (FpLibXCapture.GetMinutiaeData(varBuffer2) == TRUE) {
```

```
        // To do your code
}
}

FpLibXVerify.SecurityLevel = 5
   if (FpLibXVerify.Register(varBuffer1, varBuffer2) == TRUE) {
       // Save two minutia data to files.
}
delete[] minData1;
delete[] minData2;
```

4.7. Verification

Call the **VerifyEx()** method to match new minutiae data to the two sets of registered minutiae data. This function has three parameters: two registered minutiae data, and the minutiae data to be matched. The SecurityLevel property must be set before calling this method. For example, the security level for an application using fingerprint-only authentication can be set higher than 5 (normal) to reduce false acceptance (FAR).

- Visual Basic

```
Dim minData() as Byte
Dim savedMinData1() as Byte   ‘ Same buffer as one for registration
Dim savedMinData2() as Byte   ‘ Same buffer as two for registration

Redim minData(FpLibXCapture.MinutiaeSize) as Byte
Redim savedMinData1(FpLibXCapture.MinutiaeSize) as Byte
Redim savedMinData2(FpLibXCapture.MinutiaeSize) as Byte

If FpLibXCapture.LiveCapture(timeout, image_quality) = True then
   If FpLibXCapture.GetMinutiaeData(minData) = True then
       ‘ To do your code
   End If
Else
// Load 2 minutia data from files.(savedMinData1, savedMinData2)
FpLibXVerify.SecurityLevel = 5
If  FpLibXVerify.VerifyEx(savedMinData1, savedMinData2, minData) =
       True then
           ‘ Verification success.
Else
 ‘ Verification error.
End If
```

- C++

```
BYTE* minData = new BYTE[FpLibXCapture.MinutiaeSize];
BYTE* savedMinData1 = new BYTE[FpLibXCapture.MinutiaeSize];
BYTE* savedMinData2 = new BYTE[FpLibXCapture.MinutiaeSize];
VARIANT varCurBuffer, varBuffer1, varBuffer2;
varCurBuffer.vt = varBuffer1.vt = varBuffer2.vt = VT_BYREF|VT_UI1;
varCurBuffer.pbVal = minData;

if (FpLibXCapture.LiveCapture(timeout, image_quality) == TRUE) {
   if (FpLibXCapture.GetMinutiaeData(varBuffer1) == TRUE) {
       // To do your code
}
}
```

```
// Load 2 minutia data from files.(savedMinData1, savedMinData2)

varBuffer1.pbVal = savedMinData1;
varBuffer2.pbVal = savedMinData2;

FpLibXVerify.SecurityLevel = 5
if (FpLibXVerify.VerifyEx(varBuffer1, varBuffer2, verCurBuffer) ==
  TRUE) {
        // Verification success.
} else {
// Verification error.
}
delete[] minData;
delete[] savedMinData1;
delete[] savedMinData2;
```

The SDK provides another matching method, **Verify()** to match only one set of minutiae data. However, for best results SecuGen recommends that **VerifyEx()** be used as the method for verification processing.

Call the **Verify()** method to match minutiae data to only one registered set of minutiae data. This method has two parameters: one set of registered minutiae data, and one new set of matched data.

4.8. Matching Score

For improved quality control during the registration or verification process, you may prefer to use the matching score rather than security level to determine the success of the operation. This will allow you to accept or reject minutiae data that are registered or verified by setting a minimum score threshold instead of a security level. The matching score value may be from 0 to 199. GetMatchingScore requires two sets of minutiae data.

- Visual Basic

```
Dim min1() as Byte
Dim min2() as Byte
Dim score  as long
…
score = FpLibXVerify.GetMatchingScore(min1, min2)
If  score > 100 then
  ‘ Verification success.
Else
 ‘ Verification error.
End If
```

- C++

```
BYTE* minData = new BYTE[FpLibXCapture.MinutiaeSize];
BYTE* min1 = new BYTE[FpLibXCapture.MinutiaeSize];
BYTE* min2 = new BYTE[FpLibXCapture.MinutiaeSize];
VARIANT varCurBuffer, varBuffer1, varBuffer2;
…
varBuffer1.pbVal = min1;
varBuffer2.pbVal = min2;

DWORD score = FpLibXVerify.GetMatchingScore (varBuffer1, varBuffer2);
if (score > 100)
    // Verification success.
else
```

```
    // Verification error.
```

4.9. Selecting Template Format

FDx SDK Pro 3.5 or later supports international standard template formats as well as SecuGen's own format, which is by default. Template formats can be selected as below:

- **SecuGen's own format**

```
FpLibXVerify.MinutiaeMode = TEMPLATE_FORMAT_SG400          ' By default
FpLibXCapture.MinutiaeMode = TEMPLATE_FORMAT_SG400
```

- **ANSI378 format**

```
FpLibXVerify.MinutiaeMode = TEMPLATE_FORMAT_ANSI378        ' ANSI 378
FpLibXCapture.MinutiaeMode = TEMPLATE_FORMAT_ANSI378
```

- **ISO 19794-2 format**

```
FpLibXVerify.MinutiaeMode = TEMPLATE_FORMAT_ISO19794       ' ISO 19794-2
FpLibXCapture.MinutiaeMode = TEMPLATE_FORMAT_ISO19794
```

4.10. 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 **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 **EnableAutoOnEvent()**, pass the handle of the window which will receive the Auto-On message. The Auto-On message is defined as 0x8100 (33024 in decimals). After enabling the Auto-On message, the application writes the procedure that gets the Auto-On message from the reader. The Windows message processing routine may differ depending on the type of compiler used.

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

- **Enabling Auto-On**

```
FpLibXCapture1.EnableAutoOnEvent True, Me.hwnd
```

- **Disabling Auto-On**

```
FpLibXCapture1.EnableAutoOnEvent False, 0
```

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

```
Private g_hOldProc As Long
…

Public Function SetMonitoringDevice(hwnd As Long)
    g_hOldProc = SetWindowLong(hwnd, GWL_WNDPROC, AddressOf MyWindowProc)
End Function

Public Function StopMonitoringDevice(hwnd As Long)
    If (g_hOldProc > 0) Then
        SetWindowLong hwnd, GWL_WNDPROC, g_hOldProc
    End If
    g_hOldProc = 0
```

```
End Function

Public Function MyWindowProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long,
ByVal lParam As Long) As Long
    If (uMsg = DEVMSG_WM_DEVICE_EVENT) Then
        If (wParam = DEVMSG_DEVICE_FINGER_ON) Then
            Form1.labelErrorString.Caption = "Device event: Finger On the Sensor"
        ElseIf (wParam = DEVMSG_DEVICE_FINGER_OFF) Then
            Form1.labelErrorString.Caption = "Device event: Finger Off the Sensor"
        End If

        MyWindowProc = 0
    Else
        If (g_hOldProc) Then
            MyWindowProc = CallWindowProc(g_hOldProc, hwnd, uMsg, wParam, lParam)
        End If
    End If

End Function
```

Chapter 5. Function Reference

5.1. 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**
  pFPM
  The handle of the SGFPM object
- **Return values**
  SGFDX_ERROR_NONE = No error

5.2. 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_FDP02: device name for parallel FDP02-based readers
  SG_DEV_FDU02: device name for USB FDU02-based readers
  SG_DEV_FDU03: device name for USB FDU03 and SDU03-based readers
  SG_DEV_FDU04: device name for USB FDU04-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_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 DLL

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

5.3. 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_FDP02, SG_DEV_FDU02, 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 7.2. SGDeviceList.
devName
Device name. Should be one of the following values:
SG_DEV_UNKNOWN = 0
SG_DEV_FDP02 = 0x01
SG_DEV_FDU02 = 0x03
SG_DEV_FDU03 = 0x04
SG_DEV_FDU04 = 0x05

- **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. If the reader is a parallel type device, specify parallel port address. 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

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 7. 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. (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**
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.

5.4. 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 7.3 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

5.5. 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
  minTempate2
  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. Ifpassed 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 tempateType1, DWORD sampleNum1, BYTE* minTemplate2, WORD tempateType2, 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

5.6. 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.
  ansiTempate2
  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.
  asniTempate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  outTempate
  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**
outTempate
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
anisiTemplate
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**
ansiTempate2
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**

  ansiTempate2

  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

5.7. 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_MergeIsoTemplate()**.

- **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.
  isoTempate2
  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_MergeIsoTemplate(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_GetTIsoemplateSizeAfterMerge**() to determine the exact buffer size for **outTemplate** before calling **SGFPM_MergeIsoTemplate()**.

- **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.
  isoTempate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  outTempate
  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**

outTempate

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**

isoTempate2

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**
  isoTempate2
  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

5.8. 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 6. ActiveX Control (sgfplibx.ocx) Reference

6.1. FpLibXCapture

FpLibXCapture control includes comprehensive functions for capturing and extracting minutiae data from a fingerprint image and for drawing fingerprint data on its client window. The SecuGen fingerprint reader must be installed before FpLibXCapture control can be used.

6.1.1. Properties

CodeName
Specifies device code name

0 – FDP02 (Parallel reader based on FDP02)
1 – FDU02 (USB reader based on FDU02)
2 – FDU03 (USB reader based on FDU03 or SDU03)
3 – FDU04 (USB reader based on FDU04)

PortAddr
Specifies port address used by parallel type fingerprint reader. If selecting “0” (AUTO_DETECT), the parallel device driver will find its port address automatically. It is possible to set 0, 1, 2, or 3 when **CodeName** is set as FDP02.

0 – AUTO DETECT
1 – LPT1 (0x378)
2 – LPT2 (0x278)
3 – LPT3 (0x3BC)

DeviceID
Specifies device ID for USB type readers. **DeviceID** can have a value from 0 to 9. If **DeviceID** is -1, the USB device driver will find its port address automatically.

-1 : AUTO DETECT
0 – 9 : DeviceID

ImageWidth
Read-only width value (in pixels) of the fingerprint image

ImageHeight
Read-only height value (in pixels) of the fingerprint image

ImageSize
Buffer size of the fingerprint image. Used for allocating memory for image buffer before calling **GetImageData()** method. (Read only)

MinutiaeSize
Buffer size of minutiae data. Used for allocating memory for minutiae buffer before calling **GetMinutiaeData()** method. (Read only)

MinutiaeMode
Specifies template format to use. Used for selecting template format before calling **GetMinutiae()**. Default template format is SG400.

SG 400	SecuGen template
ANSI 378	ANSI 378 template
ISO 19794-2	ISO 19794-2 template

Brightness
Controls image brightness of the reader. Must be set to a value from 0 to 100

Contrast
Controls contrast of an image. Must be set to a value from 0 to 100

Gain
Amplifies image brightness (0-4). A higher value means a darker image. Depending on the device, this can be unavailable.

ErrorCode
Retrieves the latest error value after an error event has occurred. If **ErrorCode** is "0", there is no error. Refer to Section 8.7. SGFDxErrorCode for a complete list of error codes.

ErrorString
Gets the latest error string; used when an error event has occurred

EnableContextMenu
Enables/disables context pop-up menu on right-click of mouse button in the control's client area

IsCaptured
Checks whether an image is captured. **GetImageData** and **GetMinutiaeData** can be used only when its value is TRUE. (Read only)

Enabled
Enables/disables control

HWnd
Window handle used for displaying fingerprint image (Read only)

imageTextData
Gets a fingerprint's text image data from the buffer after capturing or setting fingerprint text image data to the buffer

MinTextData
Gets a fingerprint's text minutiae data after capturing or setting a fingerprint's text minutiae data

6.1.2. Methods

BOOL Configure()
Displays the dialog for configuring the fingerprint device

- **Parameters**
  None
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

BOOL Capture()

Captures a fingerprint image from the reader and stores it to the control buffer. Captured fingerprint is a 256 gray-level image

- **Parameters**
  None
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

BOOL LiveCapture(long Time, short Quality)

Captures fingerprint images continuously, checks the quality of the image and ignores the image if there is no fingerprint or if the image quality is unacceptable. If a quality image is captured within the given time (the second parameter), **LiveCapture**() ends its process.

- **Parameters**
  Time
  Timeout value (in milliseconds) to specify the amount of time a function will wait for a valid fingerprint to be input on the fingerprint reader
  Quality
  Image quality for accepting the fingerprint images. Must be set to a value from 0 to 100. A higher value means a higher quality fingerprint image. The default value is 50.
- **Return values**
  True = No error
  False = Failed (Refer to ErrorCode, ErrorString property)

BOOL GetImageData(const VARIANT FAR& ImageData)

Gets fingerprint image data from the buffer after capturing

- **Parameters**
  ImageData
  In Visual Basic, it is used as a buffer that is allocated by the REDIM command. With C++, use VARIANT structure with vt set to VT_BYREF | VT_UI1 and pbVal set to image buffer pointer. Buffer size is retrieved by the ImageSize property.
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

LONG GetImageQuality()

Gets quality of current captured image. A value of 60 is recommended for registration, and 40 is recommended for verification.

- **Parameters**
  None
- **Return values**
  Long = the quality value of an image

BOOL SetLed(Bool on)

Turns LED on/off in the sensor's optical unit

- **Parameters**
  on
  True or False
- **Return values**

True = No error
False = Failed (refer to ErrorCode, ErrorString property)

BOOL EnableAutoOnEvent (BOOL enable, HWND hwnd)

Allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. **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.)

- **Parameters**
  enable
  TRUE: enable Auto-On
  FALSE: disable Auto-On
  hwnd
  Window handle to receive device Auto-On message
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

BOOL GetMinutiaeData(const VARIANT FAR& MinutiaeData)

Gets a fingerprint's minutiae data after capturing. The minutiae data format is one of SG400, ANSI 378, or ISO 19794-2.

- **Parameters**
  MinutiaeData
  In Visual Basic, it is used as a buffer allocated by the REDIM command. In C++, use VARIANT structure with vt set to VT_BYREF | VT_UI1 and pbVal set to image buffer pointer. Buffer size is retrieved by MinutiaeSize property.
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

  Note: Client should allocate buffer by calling GetImageData, GetMinutiaeData

BOOL DrawImage(long hWnd)

Draws the fingerprint image in the window referenced by the window handle passed

- **Parameters**
  hWnd
  Window handle used for displaying fingerprint image
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

BOOL Refresh()

Refreshes control for redrawing an image

- **Parameters**
  None
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

BOOL Clear()

Clears client area of window and empties the buffer

- **Parameters**
  None
- **Return values**
  True = No error
  False = Failed (Refer to ErrorCode, ErrorString property)

BOOL SetImageData(const VARIANT FAR& ImageData)
Feeds **ImageData** into buffer. Calling **GetMinutiaeData**() method will then retrieve the minutiae data.

- **Parameters**
  ImageData
  In Visual Basic, used as a buffer allocated by REDIM command. With C++, use VARIANT structure with vt set to VT_BYREF | VT_UI1 and pbVal set to image buffer pointer. Buffer size is retrieved by ImageSize property.
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

long GetMatchingScore(const VARIANT FAR& MinutiaeData1, const VARIANT FAR& MinutiaeData2)
Gets matching score of two minutiae data

- **Parameters**
  MinutiaeData1
  In Visual Basic, used as a buffer allocated by REDIM command. With C++, use VARIANT structure with vt set to VT_BYREF | VT_UI1 and pbVal set to image buffer pointer. Buffer size is retrieved by ImageSize property.
  MinutiaeData2
  In Visual Basic, used as a buffer allocated by REDIM command. With C++, use VARIANT structure with vt set to VT_BYREF | VT_UI1 and pbVal set to image buffer pointer. Buffer size is retrieved by ImageSize property.
- **Return values**
  Matching score: Return value can be from 0 to 199

6.1.3. Events

Paint(OLE_HANDLE hWnd, OLE_HANDLE hDC)
Called by control to paint its client area

- **Parameters**
  hWnd
  Window handle used for displaying fingerprint image. Same as hWnd property.
  hDC
  Device context handle

Error()
Called by control when an error has occurred. To get error value, use **ErrorCode** or **ErrorString** property.

- **Parameters**
  None

6.2. FpLibXVerify

FpLibXVerify control has functions for verification and identification. FpLibXVerify control has no client area, so it is shown only during design time and hidden when the program runs. This control can be used without a SecuGen fingerprint reader installed because it is used only for matching.

6.2.1. Properties

SecurityLevel

Specifies security level for verification. Default value is NORMAL. There are nine security levels.

1 = LOWEST
2 = LOWER
3 = LOW
4 = BELOW_NORMAL
5 = NORMAL
6 = ABOVE_NORMAL
7 = HIGH
8 = HIGHER
9 = HIGHEST

ErrorCode

Retrieves the latest error value after an error event has occurred. If **ErrorCode** is “0”, there is no error. Refer to Section 8.7. SGFDxErrorCode for a complete list of error codes.

ErrorString

Gets the latest error string. Can be used when error event occurs.

DevSerialNum

Serial number of USB readers. FDU01 readers do not support DevSerialNum.

MinutiaeMode

Specifies template format to use. Used for selecting template format before calling **Verify()** or **VerifyEx()**. Default template format is SG400.

SG 400	SecuGen template
ANSI 378	ANSI 378 template
ISO 19794-2	ISO 19794-2 template

6.2.2. Methods

Bool Register(const VARIANT FAR& MinutiaeData1, const VARIANT FAR&MinutiaeData2)

Determines whether a user should be registered by comparing two sets of minutiae data. To register minutiae, the same fingerprint must be input twice on the fingerprint reader and minutiae data extracted from each image. Registration will only occur after a successful match. One or two sets of minutiae can be saved for the purpose of future verification processing.

- **Parameters**
  MinutiaeData1
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  MinutiaeData2
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
- **Return values**

True = No error
False = Failed (refer to ErrorCode, ErrorString property)

Bool Verify(const VARIANT FAR& RegMinutiaeData, const VARIANT FAR& CurMinutiaeData)

Compares a set of minutiae data and determines if the sample passes verification at the specified security level. The set of minutiae data should be in the same format, which is one of three templates such as SG 400, ANSI 378, and ISO19794-2.

- **Parameters**
  RegMinutiaeData
  Minutiae data encrypted (Compared)
  CurMinutiaeData
  Minutiae data encrypted (Comparing)
- **Return values**
  True = No error
  False = Fail (refer to ErrorCode, ErrorString property)

Bool VerifyEx(const VARIANT FAR& RegMinutiaeData1, const VARIANT FAR& RegMinutiaeData2, const VARIANT FAR& CurMinutiaeData)

Compares two sets of minutiae data and determines if the sample passes verification at the specified security level. The sets of minutiae data should be in the same format, which is one of three templates such as SG 400, ANSI 378, and ISO19794-2.

- **Parameters**
  RegMinutiaeData1, RegMinutiaeData2
  Minutiae data packet (Compared)
  CurMinutiaeData
  Minutiae data packet (Comparing)
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

Bool RegisterForText(LPCTSTR TextMinutiaeData1, LPCTSTR TextMinutiaeData2)

Determines whether a user should be registered by comparing two sets of minutiae Text data. To register minutiae, the same fingerprint must be input twice on the fingerprint reader and Text minutiae data extracted from each image. Registration will only occur after a successful match. One or two sets of minutiae can be saved for the purpose of future verification processing.

- **Parameters**
  MinutiaeData1
  Text containing minutiae data extracted from a fingerprint image
  MinutiaeData2
  Text containing minutiae data extracted from a fingerprint image
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

Bool VerifyForText(LPCTSTR TextRegMinutiaeData, LPCTSTR TextCurMinutiaeData)

Compares a set of text minutiae data and determines if the sample passes verification at the specified security level

- **Parameters**
  RegMinutiaeData
  Text minutiae data encrypted (Compared)
  CurMinutiaeData

Text Byte minutiae data encrypted (Comparing)

- **Return values**
  True = No error
  False = Fail (refer to ErrorCode, ErrorString property)

Bool VerifyEx(LPCTSTR TextRegMinutiaeData1, LPCTSTR TextRegMinutiaeData2, LPCTSTR TextCurMinutiaeData)

Compares two sets of minutiae data and determines if the sample passes verification at the specified security level

- **Parameters**
  TextRegMinutiaeData1, TextRegMinutiaeData2
  Text minutiae data packet (Compared)
  TextCurMinutiaeData
  Text Byte minutiae data packet (Comparing)
- **Return values**
  True = No error
  False = Failed (refer to ErrorCode, ErrorString property)

Bool ResetMatchingEngine()

- **Parameters**
  None
- **Return values**
  True = No error

6.2.3. Events

Error()

Called by control when an error occurs. Use **ErrorCode** or **ErrorString** properties to get the error value.

- **Parameters**
  None

Chapter 7. Structure Reference

7.1. 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**
  Contains device ID for USB readers only (0 - 9)
- **DeviceSN**
  Contains device serial number for FDU02-based USB readers or later. SGDEV_SN_LEN = 15
- **ComPort**
  Contains parallel port address for parallel readers. Contains DeviceID for USB readers.
- **ComSpeed**
  Communication speed (not used in this version)
- **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 only

7.2. 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**
  Contains device name (SG_DEV_FDP02, SG_DEV_FDU02, SG_DEV_FDU03 or SG_DEV_FDU04)
- **DevID**
  Contains device ID for USB readers only
- **DevType**
  Not used
- **DeviceSN**
  Contains device serial number for USB readers (SGDEV_SN_LEN = 15)

7.3. 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().

7.4. 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 8. Constants

8.1. SGFDxDeviceName

Device Name	Value	Description
SG_DEV_UNKNOWN	0	Not determined
SG_DEV_FDP02	0x01	FDP02-based reader
SG_DEV_FDU02	0x03	FDU02-based reader
SG_DEV_FDU03	0x04	FDU03 or SDU03-based reader
SG_DEV_FDU04	0x05	FDU04-based reader

8.2. SGPPPortAddr

Port Address	Value	Description
AUTO_DETECT	0	Auto detect
LPT1	0x378	Parallel port LPT1 address
LPT2	0x278	Parallel port LPT1 address
LPT3	0x3BC	Parallel port LPT1 address
USB_AUTO_DETECT	0x3BC+1	USB Auto detect

8.3. 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

8.4. 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

8.5. 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

8.6. 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

8.7. 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
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	Macthing filed

8.8. 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_FDU02);
```

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(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, tempateType1, sampleNum1, minTemplate2,
                                     tempateType2, sampleNum2, secuLevel, matched)
g_Fpm->GetMatchingScoreEx(minTemplate1, tempateType1, sampleNum1, minTemplate2,
                          tempateType2, 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)
```

Appendix B. Using .NET Library

To use SecuGen's .NET library, please refer to the separate manual, *FDx SDK Pro .NET Programming Manual*, for information about usage and programming.