.NET Programming Manual for FDx SDK ***Pro*** for Windows & FDx SDK ***Pro*** for Windows CE

For applications using SecuGen® fingerprint readers

SG1-0030B-008 (11/13)

Contents

Before You Begin

Biometrics Overview

Biometrics is a method of recognizing a person based on physical or behavioral characteristics. Biometric information that is used to 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” (1:N) 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” (1:1) 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 and mobile technology and other shared resources, more and more transactions of all types are initiated and completed 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 advantages over traditional 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 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 acquire one’s fingerprint with a fingerprint identification device, 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 quality optical fingerprint sensors and peripherals, software, and development kits 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 biometric community since 1998 and is an active member of the Biometrics Consortium, the International Biometrics Industry Association, and the BioAPI Consortium.

About SecuGen Products

SecuGen Sensor Qualities

- **Excellent Image Quality**: Clear, distortion-free fingerprint images are generated using advanced, patented 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

FDx SDK *Pro* provides a .NET assembly (SecuGen.FDxSDKPro.Windows.dll) for .NET developers to use SecuGen technology in a .NET framework.

Programming with SecuGen's .NET library is easy. The inclusion of fingerprint reader control, extraction, and matching algorithms in the SDK allows programmers to build biometric applications quickly and easily. All fingerprint functionality provided by the SDK is called through **SecuGen.FDxSDKPro.Windows.dll** and is accessed through the **SGFingerPrintManager** class.

1.1. System Requirements

Developer's Environment
Windows

- Windows 7, 8, 8.1 / Vista / XP / 2000, Windows Server 2012 / 2008 R2 / 2003
- .NET framework SDK 2.0 or above

Windows CE

- Microsoft Visual Studio 2005 or above
- .NET Compact Framework 1.0 or above

Run-time Environment
Windows

- Windows 7, 8, 8.1 / Vista / XP / 2000, Windows Server 2008 R2 / 2003
- .NET framework 2.0 or above

Windows CE

- Windows CE.NET 4.2, Windows CE 5.0/6.0, Windows Mobile 5.0/6.0
- .NET Compact Framework 1.0 or above

SecuGen USB readers capture and digitize fingerprint images. The host system then retrieves the image through its USB port for subsequent processing. All SecuGen USB readers, except for those based on FDU01 sensors, are supported in this SDK.

1.2. Assembly

Windows
SecuGen.FDxSDKPro.Windows.dll

Windows CE
SecuGen.FDxSDKPro.WindowsCE.dll

1.3. Namespaces

This namespace contains the SecuGen Fingerprint Management class that supports the .NET Framework

Windows
SecuGen.FDxSDKPro.Windows

Windows CE
SecuGen.FDxSDKPro.WindowsCE

1.4. Runtime files

The SecuGen .NET Library (**SecuGen.FDxSDKPro.Windows.dll or SecuGen.FDxSDKPro.WindowsCE.dll on Windows CE)** calls **sgfplib.dll**. To distribute or execute a .NET application using the SecuGen .NET Library, the following files are required.

• sgfplib.dll	Main module
• sgfpamx.dll	Fingerprint algorithm module for extraction & matching (MINEX Certified)

For more information, refer to the separate document ***FDx SDK Pro Programming Manual***.

Chapter 2. Programming

All SDK functions are implemented as members of the **SGFingerPrintManager** class. This chapter explains how to use the **SGFingerPrintManager** class to integrate SecuGen fingerprint technology into .NET applications.

2.1. Creating SGFingerPrintManager Object

In order to use the SecuGen .NET component, the **SGFingerPrintManager** object must first be instantiated. This is done by calling the **SGFingerPrintManager()** constructor.

```
[C#]
private SGFingerPrintManager m_FPM;  //member variable
…
m_FPM = new SGFingerPrintManager();

[VB.NET]
Dim m_FPM As SGFingerPrintManager 'member variable
…
m_FPM = New SGFingerPrintManager()
```

2.2. Initializing SGFingerPrintManager Object

If an **SGFingerPrintManager** object is created, it should be initialized using **Init(SGFPMDeviceName devName)** or **Init(Int32 width, Int32 height, Int32 dpi)**. **Init(SGFPMDeviceName devName)** takes the device name, loads the driver that corresponds to the device name, and initializes the fingerprint algorithm module based on device information. **Init (Int32 imageWidth, Int32 imageHeight, Int32 dpi)** takes image information to initialize fingerprint algorithm module. It does not load device driver.

- **Initiailize SGFingerPrintManage with device name**

The **Init(SGFPMDeviceName devName)** function takes a device name as a parameter. Based on the device name, the **SGFingerPrintManager** loads the required device driver module and initialize extraction module and matching module based on device information. The following table summarizes the relationships between device type, device name, loaded device driver, and initial image size when the **Init(SGFPMDeviceName devName)** function is called.

Device Type	Device Name	Value	Device Driver	Image Size (pixels)
FDP02-based	SGFPMDeviceName.DEV_FDP02	1	Parallel device driver	260*300
FDU02-based	SGFPMDeviceName.DEV_FDU02	3	USB FDU02 driver	260*300
FDU03 / SDU03-based	SGFPMDeviceName.DEV_FDU03	4	USB FDU03 / SDU03 driver	260*300
FDU04 / SDU04-based	SGFPMDeviceName.DEV_FDU04	5	USB FDU04 / SDU04 driver	258*336
U20-based	SGFPMDeviceName.DEV_FDU05	6	USB U20 driver	300*400

```
[C#]
private SGFingerPrintManager m_FPM;  //member variable
…
SGFPMDeviceName device_name = SGFPMDeviceName.DEV_FDU02;
m_FPM = new SGFingerPrintManager(device_name);

[VB.NET]
Dim m_FPM As SGFingerPrintManager 'member variable
…
Dim device_name As SGFPMDeviceName
device_name = SGFPMDeviceName.DEV_FDU02

m_FPM = New SGFINGERPRINTMANAGER(device_name)
```

- **Initiailize SGFingerPrintManager without device**

In some applications, you may need to use the **SGFingerPrintManager** class without a SecuGen reader installed on the system. In this case, you can use the overload **InitEx (Int32 imageWidth, Int32 imageHeight, Int32 dpi).** It takes image width, image height and resolution as parameters. If this function is called for initializing **SGFingerPrintManager**, the **SGFingerPrintManager** class does not load the device driver.

```
[C#]
private SGFingerPrintManager m_FPM;  //member variable
…
Int32 image_width = 260;
Int32 image_height = 300;
Int32 image_dpi = 500;

m_FPM = new SGFingerPrintManager();
err = m_FPM.InitEx(image_width, image_height, image_dpi);


[VB.NET]
Dim m_FPM As SGFingerPrintManager   'member variable
…
Dim image_width As Int32
Dim image_height As Int32
Dim image_dpi As Int32

image_width = 260
image_height = 300
image_dpi = 500

m_FPM = New SGFingerPrintManager()
err = m_FPM.Init(image_width, image_height, image_dpi)
```

2.3. Opening the SecuGen Fingerprint Reader

To use a SecuGen fingerprint reader, the reader must first be initialized by calling the **OpenDevice()** method.

The **portAddr** can have different meanings depending on which type of fingerprint reader is used.

For USB readers, **portAddr** represents the device ID. If only one USB fingerprint reader is connected to the PC, the device ID will have the value 0. If multiple USB fingerprint readers are connected to one PC, portAddr 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, **portAddr** is the actual parallel port address. Generally, if portAddr is 0 (AUTO_DETECT), the device driver will find the port address automatically.

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

USB Readers: Values used in PortAddr parameter

PortAddr	Value	Description
SGFPMPortAddr.USB_AUTO_DETECT	0x255	Detect device automatically
0 – 9	0 – 9	Device ID 0 – 9

Parallel Readers: Values used in PortAddr parameter

PortAddr	Value	Description
SGFPMPortAddr.AUTO_DETECT	0	Detect port address automatically
SGFPMPortAddr.LPT1_378	0x378	Port address 0x378
SGFPMPortAddr.LPT1_278	0x278	Port address 0x278
SGFPMPortAddr.LPT1_3BC	0x3BC	Port address 0x3BC

```
[C#]
Int32 port_addr;
…
port_addr = SGFPMPortAddr.USB_AUTO_DETECT;

iError = m_FPM.OpenDevice(port_addr);
if (iError == (Int32)SGFPMError.ERROR_NONE)
    StatusBar.Text = "Initialization Success";
else
    StatusBar.Text = "OpenDevice() Error : " + iError;

[VB.NET]
Dim port_addr As Int32

port_addr = SGFPMPortAddr.USB_AUTO_DETECT;
iError = m_FPM.OpenDevice(port_addr)

If (iError = SGFPMError.ERROR_NONE) Then
    StatusBar.Text = "Initialization Success"
Else
    StatusBar.Text = "OpenDevice() Error : " + Convert.ToString(iError)
End If
```

2.4. Getting Device Information

Device information can be retrieved by calling the **GetDeviceInfo()** method, which obtains required device information such as image height and width.

```
[C#]
SGFPMDeviceInfoParam pInfo = new SGFPMDeviceInfoParam();
pInfo = new SGFPMDeviceInfoParam ();
Int32 iError = m_FPM.GetDeviceInfo(pInfo);

if (iError == (Int32)SGFPMError.ERROR_NONE)
{
   // This should be done GetDeviceInfo();
   m_ImageWidth = pInfo.ImageWidth;
   m_ImageHeight = pInfo.ImageHeight;
}
```

```
[VB.NET]
Dim pInfo As SGFPMDeviceInfoParam
Dim iError As Int32

pInfo = New SGFPMDeviceInfoParam
iError = m_FPM.GetDeviceInfo(pInfo)

If (iError = SGFPMError.ERROR_NONE) Then
      m_ImageWidth = pInfo.ImageWidth
      m_ImageHeight = pInfo.ImageHeight
End If
```

2.5. Capturing a Fingerprint Image

After the reader is initialized, a fingerprint image can be captured using the **GetImage()** method. The captured fingerprint is a 256 gray-level image, and image width and height can be retrieved using the **GetDeviceInfo()** method. The image buffer should be allocated by the host application before calling this **GetImage()** method. There are 2 types of image capturing functions-**GetImage()** and **GetImageEx()**.

GetImage() captures an image without checking for the presence of a finger or checking image quality. **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), **GetImageEx()** ends its processing.

- **GetImage() Example**

```
[C#]
Byte[] fp_image = new Byte[m_ImageWidth*m_ImageHeight];

Int32 iError;
iError = m_FPM.GetImage(fp_image);

if (iError == (Int32)SGFPMError.ERROR_NONE)
{
   DrawImage(fp_image, pictureBox1 );
}
```

```
else
   StatusBar.Text = "GetImage() Error : " + iError;
```

[VB.NET]

```
Dim fp_image() As Byte
Dim iError As Int32

ReDim fp_image(m_ImageWidth * m_ImageHeight)
iError = m_FPM.GetImage(fp_image)

If (iError = SGFPMError.ERROR_NONE) Then
   DrawImage(fp_image, pictureBox1)
Else
   StatusBar.Text = "GetImage() Error : " + Convert.ToString(iError)
End If
```

- **GetImageEx() Example**

[C#]

```
Int32 iError;
Int32 timeout = 10000;
Int32 quality = 80;

Byte[] fp_image = new Byte[m_ImageWidth*m_ImageHeight];

iError = m_FPM.GetImageEx(fp_image, timeout, this.pictureBox1.Handle.ToInt32(), quality);
```

[VB.NET]

```
Dim fp_image() As Byte
Dim iError As Int32
Dim timeout As Int32
Dim quality As Int32

ReDim fp_image(m_ImageWidth * m_ImageHeight)

timeout = 10000
quality = 80

iError = m_FPM.GetLiveEx(fp_image, timeout, pictureBox1.Handle.ToInt32(), quality)

If (iError = SGFPMError.ERROR_NONE) Then
'
Else
   StatusBar.Text = "GetImage() Error : " + Convert.ToString(iError)
End If
```

2.6. Getting Image Quality

To determine the fingerprint image quality, use **GetImageQuality()**. **GetImageQuality** does not only check image quality and also minutiae quality.

[C#]

```
m_FPM.GetImageQuality(m_ImageWidth, m_ImageHeight, fp_image, ref img_qlty);
if (img_qlty < 80)
```

```
// Capture again
```

[VB.NET]

```
m_FPM.GetImageQuality(m_ImageWidth, m_ImageHeight, fp_image, img_qlty)
If img_qlty < 80 then
' Capture again
```

2.7. 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 controlling the brightness setting of the reader using **Configure()** or **SetBrightness()**. Using **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. **SetBrightness()** can also be used to control brightness of the reader. Brightness default values vary among the different types of SecuGen readers.

- **SetBrightness () Example**

  [C#]
  ```
  iError = m_FPM.SetBrightness(70);
  ```
  [VB.NET]
  ```
  iError = m_FPM.SetBrightness(70)
  ```

- **Configure() Example**

  [C#]
  ```
  iError = m_FPM.Configure();
  ```
  [VB.NET]
  ```
  iError = m_FPM.Configure()
  ```

2.8. 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 **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 **GetMaxTemplateSize().** It will return the maximum buffer size for data in one template. The actual template size can be obtained by calling **GetTemplateSize()** after the template is created. The **CreateTemplate()** API creates only one set of data from an image.

Note: Templates having the ANSI378 format may be merged. For more refer to Section 2.15 Manipulating ANSI378 Templates)

[C#]

```
Byte[] fp_image = new Byte[m_ImageWidth*m_ImageHeight];
Int32 iError = m_FPM.GetImage(fp_image);

iError = m_FPM.CreateTemplate(fp_image, m_RegMin1);
```

[VB.NET]

```
Dim fp_image() As Byte
```

```
ReDim fp_image(m_ImageWidth * m_ImageHeight)

iError = m_FPM.GetImage(fp_image)
iError = m_FPM.CreateTemplate (fp_image, m_RegMin1)
```

When templated is created, template information such as fingerprint position and view number can be inserted into a template. To insert a template information, use **CreateTemplate(SGFPMFingerInfo* fingerInfo, Byte rawImage[], Byte minTemplate[])**

```
[C#]
SGFPMFingerInfo finger_info = new SGFPMFingerInfo();
finger_info.FingerNumber = SGFPMFingerPosition.FINGPOS_RT;
finger_info.ImageQuality = (Int16)img_qlty;
finger_info.ImpressionType = (Int16)SGFPMImpressionType.IMPTYPE_LP;
finger_info.ViewNumber = 1;

error = m_FPM.CreateTemplate(finger_info, fp_image, m_RegMin2);
```

2.9. 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. The security level can be adjusted according to the type of application. For example, the security level for an application using fingerprint-only authentication can be set higher than **SGFPMSecurityLevel.Normal** to reduce false acceptance (FAR).

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

- **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 **SetTemplateFormat()**. For more information about template formats, refer to Section 2.14 Template Format.
- **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.
- **MatchAnsiTemplate()**: This function is the same as **MatchTemplateEx()** except that it supports only ANSI378 templates.

Function	Template Format	Can match templates with different formats?
MatchTemplate	SG400 (System default)	No
MatchTemplateEx	Specified template format	Yes
MatchAnsiTemplate	ANSI378	No
MatchIsoTemplate	ISO19794-2	No

- **MatchTemplate() Example**

[C#]
```
Int32 iError;
bool matched = false;
SGFPMSecurityLevel secu_level = SGFPMSecurityLevel.Normal;     //   Adjust   this   value
according to application type

iError = m_FPM.MatchTemplate(m_RegMin1, m_RegMin2, secu_level, ref matched);
```

[VB.NET]
```
Dim iError As Int32
Dim matched As Boolean
Dim secu_level As SGFPMSecurityLevel

secu_level = SGFPMSecurityLevel.Normal
iError = m_FPM.MatchTemplate(m_RegMin1, m_RegMin2, secu_level, matched)
```

- **MatchAnsiTemplate () Example**

[C#]
```
Int32 iError;
bool matched = false;
SGFPMSecurityLevel secu_level = SGFPMSecurityLevel.Normal;     //   Adjust   this   value
according to application type

iError = m_FPM.MatchAnsiTemplate(m_RegMin1, 0, m_RegMin2, 0, secu_level, ref matched);
```

[VB.NET]
```
Dim iError As Int32
Dim matched As Boolean
Dim secu_level As SGFPMSecurityLevel

secu_level = SGFPMSecurityLevel.Normal ‘Adjust this value according to application type

iError = m_FPM.MatchAnsiTemplate(m_RegMin1, 0, m_RegMin2, 0, secu_level, matched)
```

- **MatchTemplateEx() Example**

[C#]
```
Int32 iError;
bool matched = false;
SGFPMSecurityLevel secu_level = SGFPMSecurityLevel.Normal;     //   Adjust   this   value
according to application type

iError   =   m_FPM.MatchTemplateEx(m_RegMin1,   SGFPMTemplateFormat.SG400,   0,   m_RegMin2,
SGFPMTemplateFormat.ANSI378, 0, secu_level, ref matched);
```

[VB.NET]
```
Dim iError As Int32
Dim matched As Boolean
Dim secu_level As SGFPMSecurityLevel

secu_level = SGFPMSecurityLevel.Normal ‘Adjust this value according to application type
iError   =   m_FPM.MatchTemplateEx(m_RegMin1,   SGFPMTemplateFormat.SG400,   0,   m_RegMin2,
SGFPMTemplateFormat.ANSI378, 0, secu_level, matched)
```

2.10. 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: **GetImage()** or **GetImageEx()**
2. Extract minutiae from each captured fingerprint image: **CreateTemplate()**
3. Match each template to determine if they are acceptable for registration: **MatchTemplate()**
4. Save templates to file or database for future use

Example: Using two fingerprint images to register one fingerprint

```
[C#]
Int32 max_template_size = 0;
m_FPM.GetMaxTemplateSize(ref max_template_size);

Byte[] m_RegMin1 = new Byte[max_template_size];
Byte[] m_RegMin2 = new Byte[max_template_size];
Byte[] fp_image = new Byte[m_ImageWidth*m_ImageHeight];

// Get 1st sample
m_FPM.GetImage(fp_image);
m_FPM.CreateTemplate(fp_image, m_RegMin1);

// Get 2nd sample
iError =m_FPM.GetImage(fp_image);
iError = m_FPM.CreateTemplate(fp_image, m_RegMin2);

// Match for registration
bool matched = false;
SGFPMSecurityLevel secu_level = SGFPMSecurityLevel.Normal;

iError = m_FPM.MatchTemplate(m_RegMin1, m_RegMin2, secu_level, ref matched);

// if matched, save minutiae data to file or database
```

```
[VB.NET]
Dim max_template_size  As Int32
Dim fp_image() As Byte
Dim matched As Boolean
Dim secu_level As SGFPMSecurityLevel

ReDim fp_image(m_ImageWidth * m_ImageHeight)

‘Get 1st sample
m_FPM.GetImage(fp_image)
m_FPM.CreateTemplate(fp_image, m_RegMin1)

‘ Get 2nd sample
```

```
m_FPM.GetImage(fp_image)
m_FPM.CreateTemplate(fp_image, m_RegMin2)

'Match for registration
secu_level = SGFPMSecurityLevel.Normal

m_FPM.MatchTemplate(m_RegMin1, m_RegMin2, secu_level, matched)
```

2.11. 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: **GetImage() or GetImageEx()**
2. Extract minutiae data from captured image: **CreateTemplate()**
3. Match newly made template against registered templates:
   MatchTemplate(), MatchTemplateEx(), MatchAnsiTemplate()

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

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

[C#]

```
Int32 iError;
Byte[] fp_image = new Byte[m_ImageWidth*m_ImageHeight];
SGFPMSecurityLevel secu_level = SGFPMSecurityLevel.Normal; // Adjust this value according
to application type
bool matched1 = false;
bool matched2 = false;

//Step 1: Capture Image
m_FPM.GetImage(fp_image);

// Step 2: Create Template
m_FPM.CreateTemplate(fp_image, m_VrfMin);

// Step 3: Match for verificatation against registered template- m_RegMin1, m_RegMin2
iError = m_FPM.MatchTemplate(m_RegMin1, m_VrfMin, secu_level, ref matched1);
iError = m_FPM.MatchTemplate(m_RegMin2, m_VrfMin, secu_level, ref matched2);

if (iError == (Int32)SGFPMError.ERROR_NONE)
{
   if (matched1 & matched2)
      StatusBar.Text = "Verification Success";
   else
      StatusBar.Text = "Verification Failed";
}
else
      StatusBar.Text = "MatchTemplate() Error : " + iError;
```

```
[VB.NET]
Dim iError As Int32
Dim fp_image() As Byte
Dim matched1 As Boolean
Dim matched2 As Boolean
Dim secu_level As SGFPMSecurityLevel

ReDim fp_image(m_ImageWidth * m_ImageHeight)

'Step 1: Capture Image
iError = m_FPM.GetImage(fp_image)


'Step 2: Create Template
iError = m_FPM.CreateTemplate(fp_image, m_VrfMin)


' Step 3: Match for verificatation against registered template- m_RegMin1, m_RegMin2
secu_level = SGFPMSecurityLevel.Normal ‘Adjust this value according to application type
iError = m_FPM.MatchTemplate(m_RegMin1, m_VrfMin, secu_level, matched1)
iError = m_FPM.MatchTemplate(m_RegMin2, m_VrfMin, secu_level, matched2)

If (iError = SGFPMError.ERROR_NONE) Then
   If (matched1 And matched2) Then
      StatusBar.Text = "Verification Success"
   Else
      StatusBar.Text = "Verification Failed"
   End If

Else
   StatusBar.Text = "MatchTemplate() Error : " + Convert.ToString(iError)
End If
```

2.12. 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. **GetMatchingScore()** requires two sets of minutiae data of the same template format. **GetMatchingScoreEx()** requires two sets of minutiae data, but they can take different template formats. For more information about template formats, refer to Section 2.14 Template Format. For more information about **GetMatchingScoreEx(),** refer to Section 3.1.2. Methods - Matching Functions.

```
[C#]
Int32 match_score = 0;
m_FPM.GetMatchingScore(m_RegMin1, m_RegMin2, ref match_score);

[VB.NET]
Dim match_score As Int32

match_score = 0
```

```
        m_FPM.GetMatchingScore(m_RegMin1, m_RegMin2, match_score)
```

To understand how the matching scores correlate with typical security levels, refer to the following chart. For more information about security levels, refer to Section 3.1.2. Methods - Matching Functions.

Security Level vs. Corresponding Matching Score

SGFPMSecurityLevel	Value	Corresponding Matching Score
NONE	0	0
LOWEST	1	30
LOWER	2	50
LOW	3	60
BELOW_NORMAL	4	70
NORMAL	5	80
ABOVE_NORMAL	6	90
HIGH	7	100
HIGHER	8	120
HIGHEST	9	140

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

2.13. 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 (FDxMessage.DEV_AUTOONEVENT).

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

- **Enabling Auto-On**

```
[C#]
m_FPM.EnableAutoOnEvent(true, (int)this.Handle);
[VB.NET]
m_FPM.EnableAutoOnEvent(True, Me.Handle.ToInt32())
```

- **Disabling Auto-On**

```
[C#]
m_FPM.EnableAutoOnEvent(false, 0);

[VB.NET]
m_FPM.EnableAutoOnEvent(False, 0)
```

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

```
[C#]
protected override void WndProc(ref Message message)
{
     if (message.Msg == (int)SGFPMMessages.DEV_AUTOONEVENT)
    {
         if (message.WParam.ToInt32() == (Int32)SGFPMAutoOnEvent.FINGER_ON)
               StatusBar.Text = "Device Message: Finger On";
```

```
            else if (message.WParam.ToInt32() == (Int32)SGFPMAutoOnEvent.FINGER_OFF)
                  StatusBar.Text = "Device Message: Finger Off";
        }
        base.WndProc(ref message);
    }
```

```
[VB.NET]
Protected Overrides Sub WndProc(ByRef msg As Message)
      If (msg.Msg = SGFPMMessages.DEV_AUTOONEVENT) Then
         If (msg.WParam.ToInt32() = SGFPMAutoOnEvent.FINGER_ON) Then
            StatusBar.Text = "Device Message: Finger On"
         ElseIf (msg.WParam.ToInt32() = SGFPMAutoOnEvent.FINGER_OFF) Then
            StatusBar.Text = "Device Message: Finger Off"
         End If
      End If

      MyBase.WndProc(msg)
End Sub
```

2.14. Template Format

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

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

As default, SGFingerPrintManager creates SecuGen proprietary templates (SG400). To change the template format, use **SetTemplateFormat()**.

SG400 templates are encrypted for high security and have a size of 400 bytes. ANSI378 and ISO19794-2 templates are not encrypted, and their size is variable depending on how many fingers are 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 www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38746.

Template format

SGFPMTemplateFormat	Value	Descriptions
ANSI378	0x0100	SecuGen Proprietary template format
SG400	0x0200	ANSI-INCITS 378-2004 format
ISO19794	0x0300	ISO/IEC 19794-2:2005 format

- **Setting template format to ANSI378**

```
[C#]
m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ANSI378);
```

```
[VB.NET]
m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ANSI378);
```

- **Setting template format to ISO19794-2**

```
[C#]
m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ISO19794);

[VB.NET]
m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ISO19794);
```

- **Setting template format to SG400**

```
[C#]
m_FPM.SetTemplateFormat(m_SGFPMTemplateFormat.SG400);

[VB.NET]
m_FPM.SetTemplateFormat(m_SGFPMTemplateFormat.SG400)
```

The following functions are affected by **SetTemplateFormat()**:

- **GetMaxTemplateSize()**
- **CreateTemplate()**
- **GetTemplateSize()**
- **MatchTemplate()**
- **GetMatchingScore()**

The following APIs work only when the template format is **ANSI378**:

- **GetTemplateSizeAfterMerge()**
- **MergeAnsiTemplate()**
- **MergeMultipleAnsiTemplate()**
- **GetAnsiTemplateInfo()**
- **MatchAnsiTemplate()**
- **GetAnsiMatchingScore()**

The following APIs work with any template format:

- **MatchTemplateEx()**
- **GetMatchingScoreEx()**

2.15. Manipulating ANSI378 and ISO19794-2 Templates

The ANSI378 and ISO19794-2 template formats allow multiple fingers and multiple views per finger to be stored in a single template. To support this feature, FDx SDK *Pro* provides the following special APIs:

For ANSI378 Templates:

- **GetTemplateSizeAfterMerge()**
- **MergeAnsiTemplate()**
- **MergeMultipleAnsiTemplate()**
- **GetAnsiTemplateInfo()**
- **MatchAnsiTemplate()**
- **GetAnsiMatchingScore()**

For ISO19794-2 Templates:

- **GetIsoTemplateSizeAfterMerge()**
- **MergeIsoTemplate()**

- **MergeMultipleIsoTemplate()**
- **GetIsoTemplateInfo()**
- **MatchIsoTemplate()**
- **GetIsoMatchingScore()**

- **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 **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 **GetTemplateSizeAfterMerge()** to obtain the exact template size of the merged template before using **MergeAnsiTemplate()**.

```
[C#]
// Get first fingerprint image and create template from the image
err = GetImageEx(m_ImgBuf);
err = CreateTemplate(m_ImgBuf, m_ RegMin1);

// Get second fingerprint image and create template from the image
err = GetImageEx(m_ImgBuf);
err = CreateTemplate(m_ImgBuf, m_ RegMin2);

Byte[]  merged_template;
Int32  buf_size = 0;

m_FPM.GetTemplateSizeAfterMerge(m_RegMin1, m_RegMin2, ref buf_size);
merged_template = new Byte[buf_size];
m_FPM.MergeAnsiTemplate(m_RegMin1, m_RegMin2, merged_template);
```

- **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 **GetAnsiTemplateInfo()**.

```
[C#]
Int32 err;
SGFPMFingerPosition finger_pos = SGFPMFingerPosition.FINGPOS_UK;
bool finger_found = false;

SGFPMANSITemplateInfo sample_info = new SGFPMANSITemplateInfo();
err = m_FPM.GetAnsiTemplateInfo(m_StoredTemplate, sample_info);

for (int i = 0; i < sample_info.TotalSamples; i++)
{
   bool matched = false;
   err = m_FPM.MatchAnsiTemplate(m_StoredTemplate, i, m_VrfMin, 0, m_SecurityLevel, ref
matched);
   if (matched)
   {
      finger_found = true;
      finger_pos = (SGFPMFingerPosition)sample_info.SampleInfo[i].FingerNumber;
      break;
   }
}
```

```
if (err == (Int32)SGFPMError.ERROR_NONE)
{
   if (finger_found)
      StatusBar.Text = "The matched data found. Finger position: " + fingerpos_str[(Int32)finger_pos];
   else
      StatusBar.Text = "Cannot find a matched data";
}
else
   StatusBar.Text = "MatchAnsiTemplate() Error : " + err;

}
```

2.16. Getting Version Information of MINEX Certified Algorithm

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

```
[C#]
Int32 extractor = 0
Int32 matcher = 0;
err = m_FPM.GetMinexVersion(ref extractor, ref matcher);
```

Chapter 3. SecuGen.FDxSDKPro.Windows Reference

3.1. SGFingerPrintManager Class

Name Space: SecuGen.FDxSDKPro.Windows
Assembly Name: SecuGen.FDxSDKPro.Windows.dll

3.1.1. Constructor

SGFingerPrintManager()
Creates a new instance of the **SGFingerPrintManager** class. This constructor takes the device name or device type as an argument.

3.1.2. Methods

3.1.2.1. Initialization Functions

Int32 Init(SGFPMDeviceName deviceName)
Initializes the **SGFingerPrintManager** with deviceName. Loads device driver with device name and initializes algorithm modules based on device information.

- **Parameters**
  deviceName :
  Specifies SecuGen device name. The device name determines how the driver, extraction and matching modules are initialized.
  DEV_FDP02: device name for FDP02-based parallel port readers
  DEV_FDU02: device name for FDU02-based USB readers
  DEV_FDU03: device name for FDU03 and SDU03-based USB readers
  DEV_FDU04: device name for FDU04 and SDU04-based USB readers
  DEV_FDU05: device name for U20-based USB readers

- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_LOAD_DRIVER_MODULE = Failed to load device driver
  SGFPMError::ERROR_LOAD_EXTRACTION_MODULE = Failed to load extraction module
  SGFPMError::ERROR_LOAD_MATCHING_MODULE = Failed to load matching module

Int32 InitEx(Int32 imageWidth, Int32 imageHeight, Int32 imageDPI)
Initializes **SGFingerPrintManager** with image information. Use when running fingerprint algorithm module without a SecuGen reader.

- **Parameters**
  imageWidth:
  Image width in pixels

imageHeight:
Image height in pixels
imageDPI::
Image resolution in DPI

- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_LOAD_EXTRACTION_MODULE = Failed to load extraction module
  SGFPMError::ERROR_LOAD_MATCHING_MODULE = Failed to load matching module

Int32 SetTemplateFormat (SGFPMTemplateFormat format)
Sets template format (default is SecuGen proprietary format,**SG400**)

- **Parameters**
  Format:
  template format
  ANSI378: ANSI-INCITS 378-2004 format
  ISO19794: ISO/IEC 19794-2:2005 format
  SG400: SecuGen proprietary format
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE: Wrong template format

3.1.2.2. Device and Image Capturing functions

Int32 EnumerateDevice()
Enumerates reader(s) currently attached to the system. After calling this function, use **NumberOfDevice** property and **GetEnumDeviceInfo**() method to get enumerated reader(s).

- **Returned values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_FUNCTION_FAILED = General function fail error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter has been used

Int32 GetEnumDeviceInfo(Int32 nDevs, SGFPMDeviceList* devList)
Gets list of readers currently attached to PC. Call **GetEnumDeviceInfo()** after calling **EnumerateDevice()** method.

- **Parameters**
  ndevs
  The number of attached USB readers
  devList
  Buffer that contains device ID and device serial number. For more information, see Section 3.3 SGFPMDeviceList structure

Int32 OpenDevice(Int32 port)
Initializes the fingerprint reader

- **Parameters**
  port:
  If a USB reader is attached, the argument specifies the device ID (0 – 9). If the device ID is unknown, pass **SGFPMProtAddr::USB_AUTO_DETECT**. If parallel reader is attached, the argument specifies the parallel port address. If the port is **SGFPMPortAddr::AUTO_DETECT**, the device driver will find its port address automatically.

For USB readers, pass **SGFPMPortAddr::USB_AUTO_DETECT** or 0 – 9.
For Parallel readers, pass **SGFPMPortAddr::AUTO_DETECT, SGFPMPortAddr::LPT1_378, SGFPMPortAddr::LPT1_278** or **SGFPMPortAddr::LPT1_3BC.**

- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter was used
  SGFPMError::ERROR_SYSLOAD_FAILED = Failed to load system files
  SGFPMError::ERROR_INITIALIZE_FAILED = Failed to initialize chip
  SGFPMError::ERROR_DLLLOAD_FAILED = Failed to load module
  SGFPMError::ERROR_DEVICE_NOT_FOUND = Device not found

Int32 CloseDevice()

Closes a currently opened reader

- **Parameters**
  None
- **Return values**
  SGFPMError::ERROR_NONE = No error

Int32 Configure(int hwnd)

Displays the driver's configuration dialog box

- **Parameters**
  hwnd
  The parent window handle
- **Return values**
  SGFPMError::ERROR_NONE = No error

Int32 SetBrightness(Int32 brightness)

Controls brightness of image sensor

- **Parameters**
  brightness
  Must be set to a value from 0 to 100
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter was used

Int32 SetLedOn(bool on)

Turns optic unit LED on/off

- **Parameters**
  on
  true: Turns on LED
  false: Turns off LED
- **Return values**
  SGFPMError::ERROR_NONE = No error

Int32 GetImage(Byte buffer[]);

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

- **Parameters**

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

- **Return values**
SGFPMError::ERROR_NONE = No error
SGFPMError::ERROR_WRONG_IMAGE = Captured image is not a real fingerprint
SGFPMError::ERROR_INVALID_PARAM = An invalid parameter has been used
SGFPMError::ERROR_LINE_DROPPED = Image data is lost

Int32 GetImageQuality(Int32 width, Int32 height, Byte imgBuf[], Int32* 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 **GetImageEx().** The quality value in **GetImageEx()** represents only the ratio of the fingerprint image area to the whole scanned area.

- **Parameters**
width
Image width in pixels
height
Image height in pixels
imgBuf
Fingerprint image data
quality
The return value indicating image quality
- **Return values**
SGFPMError::ERROR_NONE = No error
SGFPMError::ERROR_INVALID_PARAM = Invalid parameter was used

Int32 GetImageEx(Byte buffer[], Int32 time, int dispWnd , Int32 quality)

Captures fingerprint images from the device 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 **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 **GetImage().** The quality value in **GetImageEx()** represents only the ratio of the fingerprint image area to the whole scanned area.

- **Parameters**
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**
SGFPMError::ERROR_NONE = No error
SGFPMError::ERROR_INVALID_PARAM = An invalid parameter has been used
SGFPMError::ERROR_LINE_DROPPED = Image data was lost

SGFPMError::ERROR_TIME_OUT = No valid fingerprint captured in the given time

Int32 EnableAutoOnEvent (bool enable, int 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 device. (Not supported by FDU02-based readers.)

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

- **Parameters**
  enable
  true: enables Auto-On
  false: disables Auto-On
  hwnd
  Window handle to receive Auto-On message
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = An invalid parameter has been 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.
  SGFPMAutoOnEvent::FINGER_ON(1) = Finger is on the sensor
  SGFPMAutoOnEvent::FINGER_OFF(0) = Finger is removed from the sensor
  lParam:
  Contains device information. The device information is contained in **SGFPMDeviceInfoParam**.

3.1.2.3. Extraction Functions

Int32 GetMaxTemplateSize(Int32* size)

Gets the maximum size of a fingerprint template (view or sample). Use this function before using **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**
  size
  The pointer to contain template size
- **Return values**
  SGFPMError::ERROR_NONE = No error

Int32 CreateTemplate(Byte rawImage[], Byte minTemplate[])

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

- **Parameters**
  rawImage
  256 Gray-level fingerprint image data
  minTemplate
  Pointer to buffer containing minutiae data extracted from a fingerprint image

- **Return values**
  SGFPMError::ERROR_NONE = No error

SGFPMError::ERROR_FEAT_NUMBER = Inadequate number of minutia
SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
SGFPMError::ERROR_INVALID_TEMPLATE1 = 103 = Error while decoding template 1
SGFPMError::ERROR_INVALID_TEMPLATE2 = 104 = Error while decoding template 2

Int32 CreateTemplate(SGFPMFingerInfo* fpInfo, Byte rawImage[], Byte minTemplate[])

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

- **Parameters**

  fpInfo

  Fingerprint information. It is stored in template. For **ANSI378** format templates, this information can be retrieved from the template using **GetAnsiTemplateInfo().** For **SG400** templates, this information cannot be seen in the template. For **SGFPMFingerInfo** structure, refer chapter 3.4.

  rawImage

  256 Gray-level fingerprint image data

  minTemplate

  Pointer to buffer containing minutiae data extracted from a fingerprint image

- **Return values**

  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_FEAT_NUMBER = Inadequate number of minutia
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = 103 = Error while decoding template 1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = 104 = Error while decoding template 2

Int32 GetTemplateSize(Byte minTemplate[], Int32* size)

Gets template size. If the template format is **SG400**, it will return 400. If the template format is **ANSI378**, template size will be varied.

- **Parameters**

  minTemplate

  Pointer to buffer containing minutiae data extracted from a fingerprint image

  size

  The pointer to contain template size

- **Return values**

  SGFPMError::ERROR_NONE = No error

3.1.2.4. Matching Functions

Int32 MatchTemplate(Byte minTemplate1[], Byte minTemplate2[], SGFPMSecurityLevel 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 **SetTemplateFormat()** and should include only one sample. To match templates that have more than one sample, use **MatchTemplateEx()** or **MatchAnsiTemplate()**. It returns **true** or **false** as a matching result (**matched**). Security levels (**secuLevel**) will affect matching result. The security level may be adjusted according to the security policy required by the user or organization.

- **Parameters**

  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

  Security level (**NORMAL** is recommended for most purposes)

LOWEST
LOWER
LOW
BELOW_NORMAL
NORMAL
ABOVE_NORMAL
HIGH
HIGHER
HIGHEST

Matched

Contains matching result. If the passed templates are the same, then **true** is returned. If not, **false** is returned.

- **Return values**

SGFPMError::ERROR_NONE = No error
SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

Int32 MatchTemplateEx(Byte minTemplate1[], SGFPMTemplateFormat tempateType1, Int32 sampleNum1, Byte minTemplate2[], SGFPMTemplateFormat tempateType2, Int32 sampleNum2, Int32 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 levels (**secuLevel**) will affect matching result. The security level may be adjusted according to the security policy required by the user or organization.

- **Parameters**

minTemplate1

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

templateType1

Specifies format of minTemplate1. Should be either **SG400 or ANSI378**.

sampleNum1

Position of a sample to be matched in minTemplate1. If templateType1 is **ANSI378**, it can have a value from 0 to the number of samples minus 1 in minTemplate1. If templateType1 is **SG400**, this value is ignored.

minTemplate2

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

templateType2

Specifies format of minTemplate2 (either **SG400 or ANSI378**)

sampleNum2

Position of a sample to be matched in minTemplate2. If templateType2 is **ANSI378**, it can have a value from 0 to the number of samples minus 1 in minTemplate2. If templateType2 is **SG400**, this value is ignored.

secuLevel

Security level (**NORMAL** is recommended for most purposes)
LOWEST
LOWER
LOW
BELOW_NORMAL
NORMAL
ABOVE_NORMAL
HIGH
HIGHER
HIGHEST

matched

Contains matching result. If the passed templates are the same, then **true** is returned. If not, **false** is returned.

- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

Int32 GetMatchingScore(Byte minTemplate1[], Byte minTemplate2[], Int32* score)

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

- **Parameters**
  minTemplate1
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  minTemplate2
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  score
  Matching score (from 0 to 199)
- **Returned values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

Int32 GetMatchingScoreEx(Byte minTemplate1[], SGFPMTemplateFormat tempateType1, Int32 sampleNum1, Byte minTemplate2[], SGFPMTemplateFormat tempateType2, Int32 sampleNum2, Int32* score);

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

- **Parameters**
  minTemplate1
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  templateType1
  Specifies format of minTemplate1 (either **SG400** or **ANSI378**)
  sampleNum1
  Position of a sample to be matched in minTemplate1. If templateType1 is **ANSI378**, it can have a value from 0 to the number of samples minus 1 in minTemplate1. If templateType1 is **SG400**, this value is ignored.
  minTemplate2
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  templateType2
  Specifies format of minTemplate2 (either **SG400** or **ANSI378**)
  sampleNum2
  Position of a sample to be matched in minTemplate2. If templateType2 is **ANSI378**, it can have a value from 0 to the number of samples minus 1 in minTemplate2. If templateType2 is **SG400**, this value is ignored.
  score
  Matching score (from 0 to 199)
- **Returned values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

3.1.2.5. Functions for ANSI378 Templates

Int32 GetTemplateSizeAfterMerge(Byte ansiTemplate1[], Byte ansiTemplate2[], Int32* size)

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

- **Parameters**
  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**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

Int32 MergeAnsiTemplate(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 **GetTemplateSizeAfterMerge**() to determine the exact buffer size for **outTemplate** before calling **MergeAnsiTemplate()**.

- **Parameters**
  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 **GetTemplateSizeAfterMerge()**.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in minTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in minTemplate2

Int32 MergeMultipleAnsiTemplate(Byte inTemplates[], Int32 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**
  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 newly merged template data. The buffer should be assigned by the application.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

Int32 GetAnsiTemplateInfo(Byte ansiTemplate[], SGFPMANSITemplateInfo* templateInfo)

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

- **Parameters**
  anisiTemplate
  ANSI378 template
  templateInfo
  The buffer that contains template information. For more information see **SGFPMANSITemplateInfo** structure.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type

Int32 MatchAnsiTemplate(Byte ansiTemplate1[], Int32 sampleNum1, Byte ansiTemplate2[], Int32 sampleNum2, SGFPMSecurityLevel secuLevel, bool* matched)

Compares two sets of ANSI378 templates. It returns **true** or **false** as a matching result (**matched**). Security levels (**secuLevel**) will affect matching result. The security level may be adjusted according to the security policy required by the user or organization.

- **Parameters**
  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 the number of samples minus 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 the number of samples minus 1 in **ansiTemplate2**.
  secuLevel
  Security level (**NORMAL** is recommended for most purposes)
  LOWEST
  LOWER
  LOW
  BELOW_NORMAL
  NORMAL
  ABOVE_NORMAL
  HIGH
  HIGHER
  HIGHEST
  matched
  Contains matching result. If the passed templates are the same, then **true** is returned. If not, **false** is returned.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in ansiTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in ansiTemplate2

Int32 GetAnsiMatchingScore(Byte ansiTemplate1[], Int32 sampleNum1, Byte ansiTemplate2[], Int32

sampleNum2, Int32*   score)

Gets matching score

- **Parameters**
  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 the number of samples minus 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 the number of samples minus 1 in **ansiTemplate2**.
  score
  Matching score (from 0 to 199)
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in ansiTemplate1
  SGFPMError.ERROR_INVALID_TEMPLATE2 = Error in ansiTemplate2

3.1.2.6. Other

Int32 GetMinexVersion(Int32 *extractor, Int32* matcher))

Gets algorithm version used in this SDK

- **Parameters**
  extractor
  MINEX compliant extractor version number
  matcher
  MINEX compliant matcher version number
- **Return values**
  SGFPMError::ERROR_NONE = No error

3.1.3. Property

Property NumberOfDevice

- **Description**
  Contains number of devices after calling **EnumerateDevice**()

3.2. SGFPMDeviceInfoParam Structure

```
public __gc struct SGFPMDeviceInfoParam
{
        Int32   DeviceID;
        Byte    DeviceSN[];
        Int32   ComPort;
        Int32   ComSpeed;
        Int32   ImageWidth;
        Int32   ImageHeight;
        Int32   Contrast;
        Int32   Brightness;
        Int32   Gain;
        Int32   ImageDPI;
        Int32   FWVersion;
};
```

Description

SGFPMDeviceInfoParam is used to obtain device information when calling GetDeviceInfo()

Constructor

SGFPMDeviceInfoParam()

Members

DeviceID: Contains device ID for USB readers only (0 –9)

DeviceSN: Contains device serial number for USB readers
Exception: FDU01-based and parallel readers do not have device serial numbers

ComPort: Contains parallel port address for parallel readers. Contains Device ID 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 value yields darker image)

ImageDPI: Image resolution of the reader in DPI

FWVersion: Device firmware version number for USB readers only

3.3. SGFPMDeviceList struct

```
public __gc struct SGFPMDeviceList
 {
        SGFPMDeviceName DevName;
        Int32           DevID;
        Int16           DevType;
        Byte            DevSN[];
 };
```

Description

Used to obtain a list of currently attached reader(s) in **GetEnumDeviceInfo()** after calling **EnumerateDevice()**

Constructor

SGFPMDeviceList()

Members

- **DevName**
  Contains device name (either SG_DEV_FDP02, SG_DEV_FDU02, SG_DEV_FDU03, SG_DEV_FDU04, or SG_DEV_FDU05)
- **DevID**
  Contains USB device ID if the device type is USB
- **DevType**
  Not used
- **DeviceSN**
  Contains device serial number of USB readers (except for FDU01-based and parallel readers). Length is defined in **DEV_SN_LEN(15)**

3.4. SGFPMFingerInfo struct

```
public __gc struct SGFPMFingerInfo
 {
       SGFPMFingerPosition        FingerNumber;
       Int16                      ViewNumber;
       Int16                      ImpressionType;
       Int16                      ImageQuality;
};
```

Description

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

Constructor

SGFPMFingerInfo();

Members

- **FingerNumber**
  Finger position number

FINGPOS_UK (0x00):	Unknown finger
FINGPOS_RT (0x01):	Right thumb

FINGPOS_RI (0x02):	Right index finger
FINGPOS_RM (0x03):	Right middle finger
FINGPOS_RR (0x04):	Right ring finger
FINGPOS_RL (0x05):	Right little finger
FINGPOS_LT (0x06):	Left thumb
FINGPOS_LI (0x07):	Left index finger
FINGPOS_LM (0x08):	Left middle finger
FINGPOS_LR (0x09):	Left ring finger
FINGPOS_LL (0x0A):	Left little finger

- **ViewNumber**
  Sample number for each finger (starts at 0)

- **ImpressionType**
  Impression type (should be 0 for SecuGen readers)

IMPTYPE_LP (0x00):	Live-scan plain
IMPTYPE_LR (0x01):	Live-scan rolled
IMPTYPE_NP (0x02):	Non-live-scan plain
IMPTYPE_NR (0x03):	Non-live-scan rolled

- **ImageQuality**
  Image quality value (0 – 100). To obtain image quality, use GetImageQuality().

3.5. SGFPMANSITemplateInfo struct

```
public __gc struct SGFPMANSITemplateInfo
 {
       Int32 TotalSamples;
       SGFPMFingerInfo* SampleInfo[];
};
```

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

Constructor
SGFPMANSITemplateInfo();

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 **SGFPMFingerInfo** structure.

3.6. SGFPMDeviceName Enumeration

Members	Value	Description
DEV_FDP02	0x01	Device code name for FDP02 parallel readers
DEV_FDU02	0x03	Device code name for FDU02 USB readers
DEV_FDU03	0x04	Device code name for FDU03 / SDU03 USB readers
DEV_FDU04	0x05	Device code name for FDU04 / SDU04 USB readers
DEV_FDU05	0x06	Device code name for U20 USB readers

3.7. SGFPMPortAddr Enumeration

Members	Value	Description
AUTO_DETECT	0	For parallel port readers: Finds port address automatically
LPT1_378	0x378	For parallel port readers: Port address is 0x378
LPT2_278	0x278	For parallel port readers: Port address is 0x278
LPT3_3BC	0x3BC	For parallel port readers: Port address is 0x3BC
USB_AUTO_DETECT	0x255	For USB readers: Finds device automatically

3.8. SGFPMSecurityLevel Enumeration

Members	Value	Description
NONE	0	No security
LOWEST	1	Lowest
LOWER	2	Lower
LOW	3	Low
BELOW_NORMAL	4	Below Normal
NORMAL	5	Normal
ABOVE_NORMAL	6	Above Normal
HIGH	7	High
HIGHER	8	Higher
HIGHEST	9	Highest

3.9. SGFPMTemplateFormat Enumeration

Members	Value	Description
ANSI378	0x0100	ANSI-INCITS 378-2004 Format
SG400	0x0200	SecuGen Proprietary Format
ISO19794	0x0300	ISO/IEC 19794-2:2005 Format

3.10. SGFPMError Enumeration

Members	Value	Description
General Error		
ERROR_NONE	0	Function success
ERROR_CREATION_FAILED	1	Failed to create SGFPM instance
ERROR_FUNCTION_FAILED	2	Function failed (various reasons)
ERROR_INVALID_PARAM	3	Invalid parameter
ERROR_NOT_USED	4	Function is not used or not supported
ERROR_DLLLOAD_FAILED	5	Cannot load device driver
ERROR_DLLLOAD_FAILED_DRV	6	Cannot load device driver
ERROR_DLLLOAD_FAILED_ALGO	7	Cannot load matching module
Device Driver Related		
ERROR_SYSLOAD_FAILED	51	Failed to load driver sys file
ERROR_INITIALIZE_FAILED = 52	52	Failed to initialize device
ERROR_LINE_DROPPED	53	Image data loss occurred during capture
ERROR_TIME_OUT	54	Timeout occurred during capture
ERROR_DEVICE_NOT_FOUND	55	Cannot find the device
ERROR_DLLLOAD_FAILED	56	Cannot load device driver
ERROR_WRONG_IMAGE	57	Wrong image – not recognized as a fingerprint image
ERROR_LACK_OF_BANDWIDTH	58	Not enough USB bandwith to complete the operation
ERROR_DEV_ALREADY_OPEN	59	Device Exclusive Access Error: cannot open the device exclusively
ERROR_GETSN_FAILED	60	Failed to get Device Serial Number
ERROR_UNSUPPORTED_DEV	61	Cannot determine device type
Extraction and Matching Related		
ERROR_FEAT_NUMBER	101	Not enough minutiae features
ERROR_INVALID_TEMPLATE_TYPE	102	Wrong template type
ERROR_INVALID_TEMPLATE1	103	Error in decoding template 1
ERROR_INVALID_TEMPLATE2	104	Error in decoding template 2
ERROR_EXTRACT_FAIL	105	Extraction failed
ERROR_MATCH_FAIL	106	Cannot find matched template

3.11. SGFPMAutoOnEvent Enumeration

Members	Value	Description
FINGER_ON	0	Finger is on the sensor
FINGER_OFF	1	Finger is off the sensor

3.12. SGFPMMessages Enumeration

Members	Value	Description
DEV_AUTOONEVENT	0x8100	Device Auto-On message (not supported by FDU02-based readers)

3.13. SGFPMImpressionType Enumeration

Members	Value	Description
IMPTYPE_LP	0x00	Live-scan plain
IMPTYPE_LR	0x01	Live-scan rolled
IMPTYPE_NP	0x02	Nonlive-scan plain
IMPTYPE_NR	0x03	Nonlive-scan rolled

3.14. SGFPMFingerPosition Enumeration

Members	Value	Description
FINGPOS_UK	0x00	Unknown finger
FINGPOS_RT	0x01	Right thumb
FINGPOS_RI	0x02	Right index finger
FINGPOS_RM	0x03	Right middle finger
FINGPOS_RR	0x04	Right ring finger
FINGPOS_RL	0x05	Right little finger
FINGPOS_LT	0x06	Left thumb
FINGPOS_LI	0x07	Left index finger
FINGPOS_LM	0x08	Left middle finger
FINGPOS_LR	0x09	Left ring finger
FINGPOS_LL	0x0A	Left little finger