SecuGen

.NET Programming Manual for FDx SDK Pro for Windows

For applications using SecuGen® fingerprint readers

SG1-0030B-018 (Last updated: 8/29/2023)

Contents

Chapter 1. Overview

FDx SDK Pro provides .NET assemblies for .NET developers to use SecuGen technology in .NET and .NET framework:

- SecuGen.FDxSDKPro.DotNet.Windows.dll for .NET 6 or higher
- SecuGen.FDxSDKPro.Windows.dll for .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 functions provided by the SDK are called through **SecuGen.FDxSDKPro.Windows.dll** or **SecuGen.FDxSDKPro.DotNet.Windows.dll** and are accessed through the **SGFingerPrintManager** class.

1.1. System Requirements[1]

Developer's Environment (Windows)
- Windows 11, 10, 8.1, 8, 7 SP1 / Vista SP2 / XP, Windows Server 2019 / 2016 / 2012 / 2008 R2 SP1
- .NET Framework SDK 4.0 or above
- .NET 6 or higher
- Visual Studio 2015 or higher recommended
  (Visual Studio 2022 for .NET 6 or higher)

Run-time Environment (Windows)
- Windows 11, 10, 8.1, 8, 7 SP1/ Vista SP2 / XP, Windows Server 2019 / 2016 / 2012 / 2008 R2 SP1
- .NET Framework 4.0 or above
- .NET 6 or higher
- Visual studio 2015 runtimes

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.

SecuGen Bluetooth Fingerprint Readers capture and digitize fingerprint images. While the host system is capable of retrieving the image wirelessly for subsequent processing, it is recommended to process the image within the Bluetooth device before transmitting to the host. This is because a fingerprint image has a larger data size compared to a fingerprint template. All SecuGen Bluetooth readers are supported in this SDK.

1.2. Assemblies

.NET: **SecuGen.FDxSDKPro.DotNet.Windows.dll**
.NET Framework: **SecuGen.FDxSDKPro.Windows.dll**

[1] U20-ASF-BT (BLE) devices will work only with Windows 10 or higher.

1.3. Namespaces

This namespace contains the SecuGen Fingerprint Management class.

SecuGen.FDxSDKPro.Windows

1.4. Runtime files

The SecuGen .NET Library[2] 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)
- sgwsqlib.dll WSQ module
- sgfdusda.dll (optional) Module for U20-ASF-BT (Bluetooth SPP and BLE) devices
- sgbledev.dll (optional) Module for U20-ASF-BT (Bluetooth BLE) devices

If a .NET application is 32-bit, the 32-bit versions of the dlls above are required, regardless of Windows systems.

Note that Visual Studio 2015 runtimes may need to be installed.

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

[2] SecuGen.FDxSDKPro.DotNet.Windows.dll or SecuGen.FDxSDKPro.Windows.dll

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

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, **SGFingerPrintManager** loads the required device driver module and initializes the extraction and matching modules based on device information. The following table summarizes the relationships among Device Type, Device Name, loaded Device Driver, and initial Image Size when the **Init(SGFPMDeviceName devName)** function is called.

Device Name	Device Sensor Type / USB Driver	Value	Image Size (pixels)
SGFPMDeviceName.DEV_FDU02	FDU02	3	260*300
SGFPMDeviceName.DEV_FDU03	FDU03 / SDU03	4	260*300
SGFPMDeviceName.DEV_FDU04	FDU04 / SDU04	5	258*336
SGFPMDeviceName.DEV_FDU05	U20	6	300*400
SGFPMDeviceName.DEV_FDU06	UPx	7	260*300
SGFPMDeviceName.DEV_FDU06AP	UPx-AP	22	300*400
SGFPMDeviceName.DEV_FDU07	U10	8	252*330
SGFPMDeviceName.DEV_FDU08	U20-A	10	300*400
SGFPMDeviceName.DEV_FDU08A	U20-AP	17	300*400
SGFPMDeviceName.DEV_FDU09A	U30	18	400*500
SGFPMDeviceName.DEV_FDU10A	U-AIR	19	500*700
SGFPMDeviceName.DEV_FDUSDA	U20-ASF-BT (SPP)	13	300*400
SGFPMDeviceName.DEV_FDUSDA_BLE	U20-ASF-BT (BLE)	14	300*400

[C#]

```
private SGFingerPrintManager m_FPM;      //member variable
…
SGFPMDeviceName device_name = SGFPMDeviceName.DEV_FDU05;
m_FPM = new SGFingerPrintManager(device_name);
```

[VB.NET]

```
Dim m_FPM As SGFingerPrintManager        'member variable
…
Dim device_name As SGFPMDeviceName
device_name = SGFPMDeviceName.DEV_FDU05

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 **InitEx2**[3] **(Int32 imageWidth, Int32 imageHeight, Int32 dpi, String *licenseFilePath).** It takes image width, image height, resolution and the path to a license file 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 = 300;
Int32 image_height = 400;
Int32 image_dpi = 500;
String pathToLicense = "license.dat";

m_FPM = new SGFingerPrintManager();
```

[3] InitEx() is no longer supported.

```
err = m_FPM.InitEx2(image_width, image_height, image_dpi, pathToLicense);
```

[VB.NET]

```
Dim m_FPM As SGFingerPrintManager        'member variable
…
Dim image_width As Int32
Dim image_height As Int32
Dim image_dpi As Int32
Dim pathToLicense As String

image_width = 300
image_height = 400
image_dpi = 500
pathToLicense = "license.dat"
m_FPM = New SGFingerPrintManager()
err = m_FPM.InitEx2(image_width, image_height, image_dpi, pathToLicense)
```

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** parameter 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.

If portAddr is **0 (AUTO_DETECT)**, the device driver will find the port address automatically.

For Serial readers such as U20-ASF-BT (SPP), portAddr is a com port.

For U20-ASF-BT (BLE) devices, the **OpenDevice(string)** method should be called with the device ID string, which can be retrieved by calling the **FindDevices()** method.

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

[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
```

For U20-ASF-BT (BLE) devices, the **FindDevices()** method can be used to get the property of devices

such as name and ID string. To cancel finding devices, the CancelFind() method can be called.

[C#]

```
void FindDevices(SGFingerPrintManager fpm) {
    uint ndevs = 0;
    uint timeout = 10000; // 10 seconds, millisecond

    int res = fpm.FindDevices(ref ndevs, timeout);
    Assert.IsTrue(res == (int)SGFPMError.ERROR_NONE);

    Console.WriteLine("{0} devic(s) found.", ndevs);
    if (ndevs > 0) {
        SGFPMDeviceInfo devInfo = new SGFPMDeviceInfo();
        for (int i = 0; i != ndevs; i++) {
            res = fpm.GetDeviceInfoFound(i, devInfo);
            string name = new string(devInfo.ID);
            string id = new string(devInfo.Name);
            Console.WriteLine("Name: {0},  ID: {1}", name, id);
       }
    }
}
```

2.5. Capturing a Fingerprint Image[4]

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 one image without additional requirements. But If **GetImage()** needs to be called multiple times, it is recommended to call **BeginGetImage()** before and **EndGetImage()** after the series of calls for **GetImage()**. This will allow **GetImage()** to run faster if the device, such as U-Air, supports these two functions. For more information, please review the MatchingUAIR sample code.

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)
{
```

[4] For Bluetooth devices, the method **CreateTemplateDev()** is preferred over capturing images because it can take a long time to wirelessly transmit an image (large data size) compared to a template (small data size).

```
        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
```

- **CreateTemplateDev() Example (U20-ASF-BT only)**

[C#]

```
    void CreateTemplateDev() {
         // see what template format is.
```

```
        int templateFormat = 0;
        int res = _fpm.GetTemplateFormatDev(ref templateFormat);
        Assert.IsTrue(res == (int)SGFPMError.ERROR_NONE);

        // capture a fingerprint and create a template
        int sizeTemplate = 0;
        res = _fpm.CreateTemplateDev(ref sizeTemplate);
        Assert.IsTrue(res == (int)SGFPMError.ERROR_NONE);

        // get a template
        byte[] min = new byte[sizeTemplate];
        res = _fpm.GetTemplateDev(min);
        Assert.IsTrue(res == (int)SGFPMError.ERROR_NONE);

        // save it
        SaveTemplate(min, templateFormat);
    }
```

2.6. Getting Image Quality

To determine the fingerprint image quality, you can use **GetImageQuality()**. **GetImageQuality** checks both image quality and minutiae quality. Alternatively, **GetLastImageQuality()**[5] can be used as a simple and fast way to check the quality of the last image captured from the device.

[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);
```

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

```
[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[6]

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, ISO19794-2, or ISO19794-2 compact[7] card formats may be merged. For more information about template formats and merging formats, refer to the following Sections:

Section 2.14 Template Format
Section 2.15 Manipulating ANSI378, ISO19794-2, and ISO19794-2 Compact 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 a 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#]
```

[6] For Bluetooth devices, the **CreateTemplateDev()** method is preferred over **GetImage()**because it can take a long time to wirelessly transmit an image (large data size) compared to a template (small data size).
[7] For Bluetooth devices, the ISO19794-2 compact format is not supported.

```
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 = 0;

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 four 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.
- **MatchIsoCompactTemplate()**: This function is the same as **MatchTemplateEx()** except that it supports only ISO19794-2 compact card 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
MatchIsoCompactTemplate	ISO19794-2 compact format	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(), MatchIsoTemplate(),** or **MatchIsoCompactTemplate()**

   - Adjust the security level according to the type of application. For example, if fingerprint-only authentication 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 3.1.2.4 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 score correlates with typical security levels, refer to the following chart. For more information about security levels, refer to Section 3.1.2.4 Matching Functions.

Security Level vs. Matching Score

SGFPMSecurityLevel	Value	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 3.81 of FDx SDK Pro, the 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 four types of fingerprint template formats:

- **SG400:** SecuGen's proprietary template format
- **ANSI378:** ANSI-INCITS 378-2004 "Finger Minutiae Format for Data Exchange"
- **ISO19794-2:** ISO/IEC 19794-2:2005 "Biometric Data Interchange Formats – Part 2: Finger Minutiae Data"
- **ISO19794-2 Compact**[8]**:** ISO/IEC 19794-2:2005 "Biometric Data Interchange Formats– Part 2: Finger Minutiae Data" – Section 8.2 Compact Size Finger Minutiae Format (Compact Card Format)

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

For U20-ASF-BT devices, the **SetTemplateFormatDev()** method can be used to change the format for **CreateTemplateDev()**. Note that the **SetTemplateFormat()** will not affect **CreateTemplateDev()**.

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.

[8] For U20-ASF-BT devices, this format is not supported.

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

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

Template format

SGFPMTemplateFormat	Value	Description
ANSI378	0x0100	ANSI-INCITS 378-2004 format
SG400	0x0200	SecuGen Proprietary template format
ISO19794	0x0300	ISO/IEC 19794-2:2005 format
ISO19794_COMPACT	0x0400	ISO/IEC 19794-2:2005 compact card 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 ISO19794-2 compact card format**

```
[C#]
	m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ISO19794_COMPACT);

[VB.NET]
	m_FPM.SetTemplateFormat(SGFPMTemplateFormat.ISO19794_COMPACT);
```

- **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 only when the template format is **ISO19794**:

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

The following APIs work only when the template format is **ISO19794_COMPACT**:

- **GetIsoCompactTemplateSizeAfterMerge()**
- **MergeIsoCompactTemplate()**
- **MergeMultipleIsoCompactTemplate()**
- **GetIsoCompactTemplateInfo()**
- **MatchIsoCompactTemplate()**
- **GetIsoCompactMatchingScore()**

The following APIs work with any template format:

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

2.15. Manipulating ANSI378, ISO19794-2, and ISO19794-2 Compact Templates

The ANSI378, ISO19794-2, and ISO19794-2 Compact 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()**

For ISO19794-2 Compact Card Templates:

- **GetIsoCompactTemplateSizeAfterMerge()**
- **MergeIsoCompactTemplate()**
- **MergeMultipleIsoCompactTemplate()**
- **GetIsoCompactTemplateInfo()**
- **MatchIsoCompactTemplate()**
- **GetIsoCompactMatchingScore()**

- **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 size of the merged template will be smaller 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.DotNet.Windows.dll or 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_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
  DEV_FDU06: device name for UPx-based USB readers
  DEV_FDU06AP: device name for UPx-AP based USB readers
  DEV_FDU07: device name for U10-based USB readers
  DEV_FDU08: device name for U20-A based USB readers
  DEV_FDU08A: device name for U20-AP based USB readers
  DEV_FDU09A: device name for U30 based USB readers
  DEV_FDU10A: device name for U-AIR based contactless USB readers
  DEV_FDUSDA: device name for U20-ASF-BT (Bluetooth SPP) based readers
  DEV_FDUSDA_BLE: device name for U20-ASF-BT (Bluetooth BLE) based readers
  DEV_AUTO: device name for any devices above

- **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 InitEx2[9](Int32 imageWidth, Int32 imageHeight, Int32 imageDPI, String pathToLicense)

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
  pathToLicense:
  Path to a license file
- **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
  ISO19794_COMPACT: ISO/IEC 19794-2:2005 compact card 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

[9] InitEx() is no longer supported

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

int FindDevices(ref uint ndevs, uint timeout)

Find U20-ASF-BT (BLE) devices.

- **Parameters**

  ndevs:

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

  timeout:

  Timeout in millisecond

- **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

int GetDeviceInfoFound(int ndev, SGFPMDeviceInfo devInfo)

Find the info of the U20-ASF-BT (BLE) device found by calling **FindDevices()**.

- **Parameters**

  ndev:

  The index of the U20-ASF-BT (BLE) device found to get the info of.

  devinfo:

  A buffer where the info will be written.

- **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

int CanelFind()

Cancel finding U20-ASF-BT (BLE) devices. The **FindDevices()** method will stop and return immediately.

- **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

Int32 OpenDevice(Int32 port)

Initializes the fingerprint reader

- **Parameters**

  port:

  If a USB reader is attached, the argument specifies the device ID (from 0 to 9). If the device ID is unknown, pass **SGFPMProtAddr::USB_AUTO_DETECT**. 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.

- **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 OpenDevice(string id)

Initializes the U20-ASF-BT (BLE) reader

- **Parameters**
  id:
  A ID string from the **FindDevices()** method. For instance, it will have like in c#:
  `string id = "BluetoothLE#BluetoothLEac:d1:b8:d0:d6:e4-cc:35:5a:ff:f0:37";`
- **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
  Brightness 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 BeginGetImage()

Prepares for **GetImage**() to start capture. Must call **EndGetImage()** later.

- **Parameters**
  None
- **Return values**
    SGFPMError::ERROR_NONE = No error
    SGFPMError::ERROR_INVALID_PARAM = An invalid parameter has been used
    SGFPMError::ERROR_UNSUPPORTED_DEV = Not supported

Int32 EndGetImage()

Closes for **GetImage**() to finish capture

- **Parameters**
  None
- **Return values**
    SGFPMError::ERROR_NONE = No error
    SGFPMError::ERROR_INVALID_PARAM = An invalid parameter has been used
    SGFPMError::ERROR_UNSUPPORTED_DEV = Not supported

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 GetLastImageQuality(ref int quality)

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

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

- **Parameters**
  quality
  The return value indicating image quality
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_PARAM = Invalid parameter was used
  SGFPMError::ERROR_UNSUPPORTED_DEV = Not supported

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 a fixed length size of 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 stored in a template. For **ANSI378** templates, this information can be retrieved from the template using **GetAnsiTemplateInfo()**. For **ISO19794** templates, this information can be retrieved from the template using **GetIsoTemplateInfo()**. For **ISO19794_COMPACT** templates, this information can be retrieved from the template using **GetIsoCompactTemplateInfo().** For **SG400** templates, this information cannot be seen in the template. For more information, refer to Section 3.4 SGFPMFingerInfo Structure.

  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

int CreateTemplateDev(ref int size)

Captures a fingerprint image and extracts minutiae to form a template having the default format. To get the template, call **GetTemplateDev()** with a buffer having the size. (U20-ASF-BT only)

- **Parameters**

  size

  The size of the template created

- **Return values**

  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_FUNCTION_FAILED

int GetTemplateDev(byte[] min)

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

- **Parameters**

  min

  An array of bytes with the size of the template returned by **CreateTemplateDev()**.

- **Return values**

  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_FUNCTION_FAILED

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, ISO19794, or ISO19794_COMPACT**, 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()**, **MatchAnsiTemplate()**, **MatchIsoTemplate(),** or **MatchIsoCompactTemplate**. It returns **true** or **false** as a matching result (**matched**). The security level (**secuLevel**) will affect matching result and 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
  minTemplate2
  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 templateType2, Int32 sampleNum2, Int32 secuLevel, bool* matched)

Compares two sets of minutiae data, which can be of different template formats (SG400, ANSI378, ISO19794, or ISO19794_COMPACT). It returns **true** or **false** as a matching result (**matched**). The security level (**secuLevel**) will affect matching result and 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 (**SG400, ANSI378, ISO19794, or ISO19794_COMPACT**)
  sampleNum1
  Position of a sample to be matched in minTemplate1. If templateType1 is **ANSI378, ISO19794, or ISO19794_COMPACT**, 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 (**SG400, ANSI378, ISO19794, or ISO19794_COMPACT**)
  sampleNum2
  Position of a sample to be matched in minTemplate2. If templateType2 is **ANSI378, ISO19794, or ISO19794_COMPACT**, 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 templateType1, Int32 sampleNum1, Byte minTemplate2[], SGFPMTemplateFormat templateType2, Int32 sampleNum2, Int32* score)

Gets matching score of two sets of minutiae data, which can be of different template formats (SG400, ANSI378, ISO19794, or ISO19794_COMPACT)

- **Parameters**
  minTemplate1
  A pointer to the buffer containing minutiae data extracted from a fingerprint image
  templateType1
  Specifies format of minTemplate1 (**SG400, ANSI378, ISO19794, or ISO19794_COMPACT**)
  sampleNum1
  Position of a sample to be matched in minTemplate1. If templateType1 is **ANSI378, ISO19794, or ISO19794_COMPACT**, 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 (**SG400, ANSI378, ISO19794, or ISO19794_COMPACT**)
  sampleNum2
  Position of a sample to be matched in minTemplate2. If templateType2 is **ANSI378, ISO19794, or ISO19794_COMPACT**, 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.
  ansiTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  size
  Template size if two templates are merged
- **Return values**
  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 MergeAnsiTemplate(Byte ansiTemplate1[], Byte ansiTemplate2[], Byte outTemplate[])
Merges two ANSI378 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than 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.
  asniTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  outTemplate
  The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **GetTemplateSizeAfterMerge()**.
- **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 MergeMultipleAnsiTemplate(Byte inTemplates[], Int32 nTemplates, Byte outTemplate[])
Merges multiple ANSI378 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of all templates in **inTemplates**.

- **Parameters**
  inTemplates
  A series of ANSI378 templates [ANSITemplate-1, ANSITemplate-2, ANSITemplate-3, ... ANSITemplate-n]
  nTemplates
  The number of templates in **inTemplates**
  outTemplate
  The buffer containing 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**
  ansiTemplate
  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**). The security level (**secuLevel**) will affect matching result and 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**.

  ansiTemplate2

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

  sampleNum2

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

  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**.

  ansiTemplate2

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

  sampleNum2

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

  score

  Matching score (from 0 to 199)

- **Return values**
  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. Functions for ISO19794 Templates

Int32 GetIsoTemplateSizeAfterMerge(Byte isoTemplate1[], Byte isoTemplate2[], Int32* size)
Calculates template size if two templates – isoTemplate1 and isoTemplate2 – are merged. Use this function to determine the exact buffer size before using **MergeIsoTemplate()**.

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  size
  Template size if two templates are merged
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 MergeIsoTemplate(Byte isoTemplate1[], Byte isoTemplate2[], Byte outTemplate[])
Merges two ISO19794 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of the two input templates (size of ansiTemplate1 + size of ansiTemplate2). Call **GetIsoTemplateSizeAfterMerge**() to determine the exact buffer size for **outTemplate** before calling **MergeIsoTemplate()**.

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  outTemplate
  The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **GetIsoTemplateSizeAfterMerge()**.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 MergeMultipleIsoTemplate(Byte inTemplates[], Int32 nTemplates, Byte outTemplate[])
Merges multiple ISO19794 templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of all templates in **inTemplates**.

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

nTemplates
The number of templates in **inTemplates**
outTemplate
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 GetIsoTemplateInfo(Byte isoTemplate[], SGFPMANSITemplateInfo* templateInfo)
Gets information of an ISO19794 template. Call this function before **MatchIsoTemplate()** to obtain information about a template.

- **Parameters**
  isoTemplate
  ISO19794 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 MatchIsoTemplate(Byte isoTemplate1[], Int32 sampleNum1, Byte isoTemplate2[], Int32 sampleNum2, SGFPMSecurityLevel secuLevel, bool* matched)
Compares two sets of ISO19794 templates. It returns **true** or **false** as a matching result (**matched**). The security level (**secuLevel**) will affect matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  sampleNum1
  Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  sampleNum2
  Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**.
  secuLevel
  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 isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 GetIsoMatchingScore(Byte isoTemplate1[], Int32 sampleNum1, Byte isoTemplate2[], Int32 sampleNum2, Int32* score)

Gets matching score

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  sampleNum1
  Position of sample to be matched in **isoTemplate1**. It can be from 0 to the number of samples minus 1 in **isoTemplate1**.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  sampleNum2
  Position of sample to be matched in **isoTemplate2**. It can be from 0 to the number of samples minus 1 in **isoTemplate2**.
  score
  Matching score (from 0 to 199)
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

3.1.2.7. Functions for ISO19794 Compact Card Templates

Int32 GetIsoCompactTemplateSizeAfterMerge(Byte isoTemplate1[], Byte isoTemplate2[], Int32* size)

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

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  size
  Template size if two templates are merged
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 MergeIsoCompactTemplate(Byte isoTemplate1[], Byte isoTemplate2[], Byte outTemplate[])

Merges two ISO19794 compact card templates and returns a new merged template. The size of the merged template (**outTemplate**) will be smaller than the sum of the sizes of the two input templates (size of ansiTemplate1 + size of ansiTemplate2). Call **GetIsoCompactTemplateSizeAfterMerge()** to determine the exact buffer size for **outTemplate** before calling **MergeIsoCompactTemplate()**.

- **Parameters**
  isoTemplate1
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  isoTemplate2
  A pointer to the buffer containing minutiae data. A template can have more than one sample.
  outTemplate
  The buffer containing merged data. The buffer should be assigned by the application. To determine the exact buffer size, call **GetIsoCompactTemplateSizeAfterMerge()**.
- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 MergeMultipleIsoCompactTemplate(Byte inTemplates[], Int32 nTemplates, Byte outTemplate[])

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

- **Parameters**
  inTemplates
  A series of ISO19794 compact card templates [ISOTemplate-1, ISOTemplate-2, ISOTemplate-3, ... ISOTemplate-n]
  nTemplates
  The number of templates in **inTemplates**
  outTemplate
  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 GetIsoCompactTemplateInfo(Byte isoTemplate[], SGFPMANSITemplateInfo* templateInfo)

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

- **Parameters**
  isoTemplate
  ISO19794 compact card 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 MatchIsoCompactTemplate(Byte isoTemplate1[], Int32 sampleNum1, Byte isoTemplate2[], Int32 sampleNum2, SGFPMSecurityLevel secuLevel, bool* matched)

Compares two sets of ISO19794 compact card templates. It returns **true** or **false** as a matching result (**matched**). The security level (**secuLevel**) will affect matching result and may be adjusted according to the security policy required by the user or organization.

- **Parameters**

  isoTemplate1

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

  sampleNum1

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

  isoTemplate2

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

  sampleNum2

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

  secuLevel

  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 isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

Int32 GetIsoCompactMatchingScore(Byte isoTemplate1[], Int32 sampleNum1, Byte isoTemplate2[], Int32 sampleNum2, Int32* score)

Gets matching score

- **Parameters**

  isoTemplate1

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

  sampleNum1

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

  isoTemplate2

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

  sampleNum2

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

score
Matching score (from 0 to 199)

- **Return values**
  SGFPMError::ERROR_NONE = No error
  SGFPMError::ERROR_INVALID_TEMPLATE_TYPE = Wrong template type
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2
  SGFPMError::ERROR_INVALID_TEMPLATE1 = Error in isoTemplate1
  SGFPMError::ERROR_INVALID_TEMPLATE2 = Error in isoTemplate2

3.1.2.8. 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 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
ComPort	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

3.3. SGFPMDeviceInfo Structure

```
public class SGFPMDeviceInfo {
        public char[] ID;
        public char[] Name;
}
```

Description

Used to obtain the propereties of a U20-ASF-BT (BLE) reader in **GetDeviceInfoFound()** after calling **FindDevices()**

Constructor

SGFPMDeviceList()

Members

ID

Contains U20-ASF-BT (BLE) device ID

DevName

Contains device name

3.4. SGFPMDeviceList Structure

```
Public 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 (SG_DEV_FDU02, SG_DEV_FDU03, SG_DEV_FDU04, SG_DEV_FDU05, SG_DEV_FDU06, SG_DEV_FDU06AP, SG_DEV_FDU07, SG_DEV_FDU08, SG_DEV_FDU08A, SG_DEV_FDU09A, SG_DEV_FDU10A, SG_DEV_FDUSDA, SG_DEV_FDUSDA_BLE)

DevID

Contains USB device ID if the device type is USB

DevType

Not used

DeviceSN

Contains device serial number of USB readers. Length is defined in **DEV_SN_LEN(15)**

3.5. SGFPMFingerInfo Structure

```
public 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, ISO19794-2,** and **ISO19794-2 Compact** 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	Finger
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()**.

3.6. SGFPMANSITemplateInfo Structure

```
public_struct SGFPMANSITemplateInfo
 {
        Int32 TotalSamples;
        SGFPMFingerInfo* SampleInfo[];
};
```

Description

Used when calling **GetAnsiTemplateInfo ().** The provided information will be put into the template. For **ANSI378, ISO19794-2,** and **ISO19794-2 Compact** 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 section 3.4 SGFPMFingerInfo Structure.

3.7. SGFPMDeviceName Enumeration

Members	Value	Description
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
DEV_FDU06	0x07	Device code name for UPx USB readers
DEV_FDU06AP	0x16	Device code name for UPx-AP USB readers
DEV_FDU07	0x08	Device code name for U10 USB readers
DEV_FDU08	0x0A	Device code name for U20-A USB readers
DEV_FDU08A	0x11	Device code name for U20-AP USB readers
DEV_FDU09A	0x12	Device code name for U30 USB readers
DEV_FDU10A	0x13	Device code name for U-Air USB readers
DEV_FDUSDA	0x0D	Device code name for U20-ASF-BT (Bluetooth SPP) readers
DEV_FDUSDA_BLE	0x0E	Device code name for U20-ASF-BT (Bluetooth BLE) readers

3.8. SGFPMPortAddr Enumeration

Members	Value	Description
USB_AUTO_DETECT	0x255	Finds USB device automatically

3.9. 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

3.10. 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
ISO19794_COMPACT	0x0400	ISO/IEC 19794-2:2005 Compact Card Format

3.11. 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
ERROR_NO_LONGER_SUPPORTED	8	No longer supported
ERROR_DLLLOAD_FAILED_WSQ	9	Sgwsqlib.dll not loaded
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
ERROR_FAKE_FINGER	62	Fake finger detected
ERROR_FAKE_INITIALIZED	63	Initialization of fake module failed
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
License Related		
ERROR_LICENSE_LOAD	501	License file not found
ERROR_LICENSE_KEY	502	License key is invalid
ERROR_LICENSE_EXPIRED	503	License expired
WSQ Related		
ERROR_NO_IMAGE	600	Invalid image

3.12. SGFPMAutoOnEvent Enumeration

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

3.13. SGFPMMessages Enumeration

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

3.14. 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

3.15. 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