SecuSearch® SDK *Pro* Manual
Windows

For one-to-many fingerprint identification applications
Using SecuGen® fingerprint readers

SG1-0005C-003 (06/14)

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

About SecuGen

SecuGen (www.secugen.com) provides biometric solutions for physical and network security employing advanced fingerprint recognition technology. The company’s comprehensive product line includes high quality optical fingerprint readers and sensor component, software and development kits that are used for a variety of innovative applications including Internet, enterprise network and desktop security, physical access control, time and attendance management and financial and medical records control. SecuGen patented products feature the industry’s longest warranty and are renowned for their accuracy, reliability and versatility. Based in Silicon Valley, SecuGen has been serving the global biometric community since 1998 and is an active member of the Biometrics Consortium, 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 and patent-pending optical methods. Quality imaging yields better sampling for minutiae data extraction.
- **Durability:** Mechanical strength tests show resistance to impact, shock and scratches.
- **Powerful Software:** Precise, fast processing algorithm ensures efficiency and reliability.
- **Ruggedness and Versatility:** Solid engineering and superior materials allow for use under extreme conditions.
- **Ergonomic Design:** Compact, modular design for seamless integration into small devices, ease of use, and compatibility make SecuGen sensors 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 SecuGen sensors 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

The SecuSearch® SDK *Pro* is the one-to-many (1:N) matching engine software developer's kit that enables programmers to develop extremely fast, highly accurate fingerprint searching programs for use in large-scale fingerprint databases.

The SecuSearch SDK can be used for two typical classes of applications:

- To identify unknown individuals by matching fingerprints in a fingerprint database (e.g., searching for missing children, criminal investigations, etc.)
- To replace identification codes with a high security, user-friendly method (e.g., time and attendance systems, member management systems, system login without ID)

The SecuSearch SDK supports quick and easy 1:N matching system integration in any fingerprint database application where accuracy and search speed are paramount.

1.1. Features of SecuSearch SDK

- **Optimized API**
  Offers an optimized API for fingerprint registration and searching so that programmers can quickly and easily build fingerprint search systems
- **High-speed fingerprint searching**
  Utilizes an innovative indexing-based algorithm that is different from sequential comparison and that increases the search speed to over 10,000 fingerprints per second in a PC environment
- **High accuracy in fingerprint matching**
  Provides accurate candidate lists with corresponding confidence levels
- **Wide compatibility**
  Can be easily integrated into any type of platform since all codes are designed in the ANSI C environment and currently supports Windows 7/Vista/XP/2000/NT/2003/2008.

1.2. Embedded Functions

SecuSearch Engine uses the fingerprint template data extracted by SecuGen Fingerprint APIs. Registration and searching use the fingerprint template obtained from the image captured by SecuGen's fingerprint reader.

Main functions of SecuSearch Engine

Begin / End / Parameter Setting

- **SecuSrchAPI_InitializeEngine**
  Initializes the Fingerprint Database, allocates database memory and initializes global variables.
- **SecuSrchAPI_TerminateEngine**
  Clears the Fingerprint Database, deletes all fingerprint information located in the database and frees allocated memory
- **SecuSrchAPI_SetEngineParam**
  Sets the parameters of the SecuSearch Engine

- **SecuSrchAPI_GetEngineParam**
  Retrieves values of SecuSearch Engine parameters

Fingerprint Registration / Deletion / Searching

- **SecuSrchAPI_RegisterFP**
  Registers new fingerprints into the database
- **SecuSrchAPI_SearchFP**
  Searches the fingerprints in the database and outputs a list of matching candidates. Matching candidates are the fingerprints in the database that show similarity to the query fingerprint.
- **SecuSrchAPI_IdentifyFP**
  Identifies the fingerprints in the database and outputs the fingerprint most similar to the query fingerprint, that is, a fingerprint in the database with a similarity value higher than the given level and threshold
- **SecuSrchAPI_Remove**
  Deletes the fingerprint data stored in the database
- **SecuSrchAPI_ClearFPDB**
  Deletes all fingerprint data from the memory resident fingerprint database
- **SecuSrchAPI_GetFPCount**
  Retrieves number of templates registered in the memory resident fingerprint database
- **SecuSrchAPI_GetFPList**
  Retrieves a list of templates registered in the memory resident fingerprint database
- **SecuSrchAPI_GetFPData**
  Retrieves template data registered in the memory resident fingerprint database

Saving / Loading of Database (Not recommended for the latest version)

- **SecuSrchAPI_SaveFPDB**
  Saves a memory resident fingerprint database to the disk
- **SecuSrchAPI_LoadFPDB**
  Loads a fingerprint database file from the disk to memory

Chapter 2. Installation

2.1. System requirements

OS	Windows 8.1 / 8 / 7 / Vista / XP / 2000 / NT
	Windows Server 2012 / 2008 R2 / 2008 / 2003
CPU	Pentium 2GHz or higher
System Memory	2GB or larger
Library	SecuGen fingerprint API
For .NET Development	
	.NET Framework version 2.0 or higher

The SecuSearch Engine is designed to operate with SecuGen fingerprint readers and SecuGen fingerprint feature-extraction algorithms provided in the FDx SDK *Pro*, SecuBSP® SDK *Pro* or SDA0x/FDA0x Developer Kits. Such kits, depending on your choice of reader and/or interface, must be must be installed before beginning any application design.

The SecuGen fingerprint readers that work with SecuSearch include:

- All USB and parallel PC peripherals
- All SDA0x and FDA0x stand-alone sensors with built-in CPU

Memory requirements of the SecuSearch Engine vary according to the number of registered fingerprints because the fingerprint database resides on the system's main memory (RAM).

Please note that the whole amount of required memory must be continuous. In order to have a large amount of continuous memory block, the system memory might have to be much larger. For example, for 500M of continuous memory, at least 3GB system memory is recommended. Running other applications may reduce the maximum size of continuous memory available.

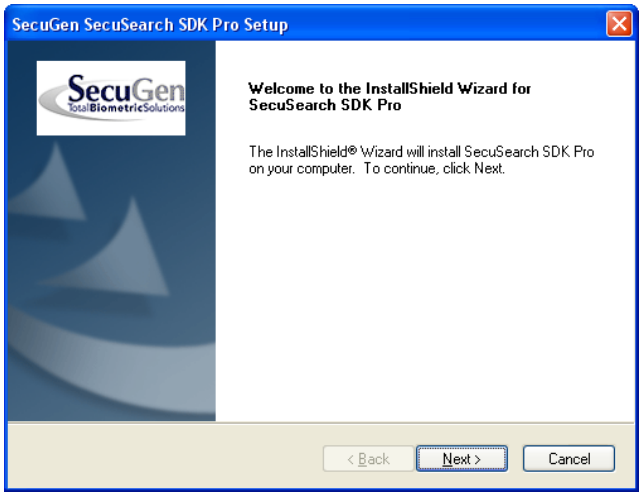
Number of Registered Fingerprints	Required Memory Size
100	10 Mbytes
1,000	50 Mbytes
2,000	80 Mbytes
5,000	160 Mbytes
10,000	250 Mbytes
20,000	400 Mbytes
30,000	500 Mbytes

Software requirements

To execute some of the sample programs, MS Access database is required.

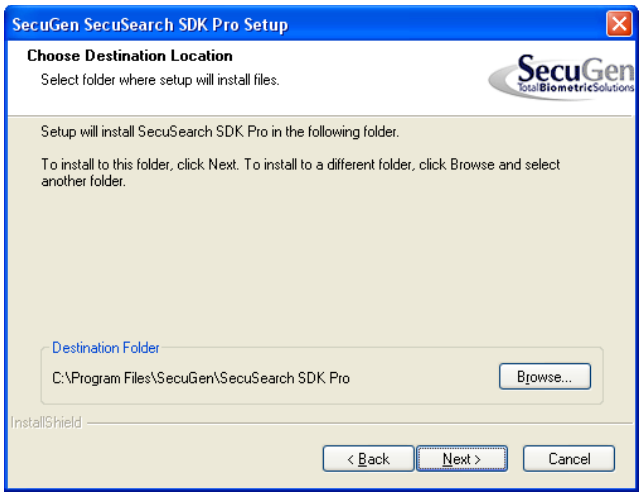
2.2. SecuSearch Engine Installation

1. Insert the SecuGen SDK Collection CD into the CD-ROM drive.
2. Run setup.exe from the “SecuSearch SDK Pro” root directory.
3. Click **Next** when the setup screen is shown.

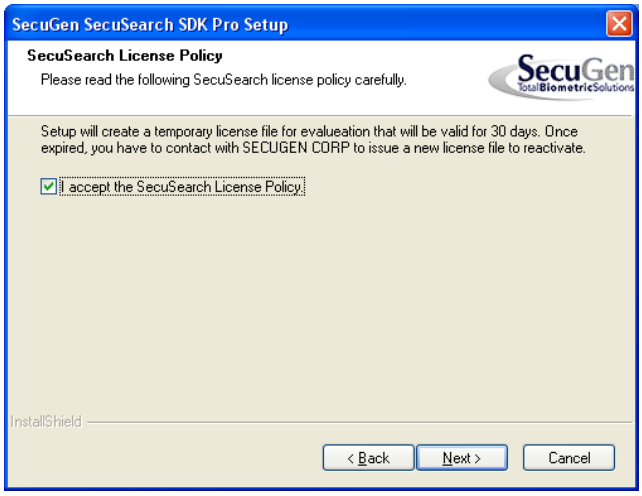


4. Click **Yes** after reading the license agreement to continue, or **No** to cancel the installation if you do not agree.

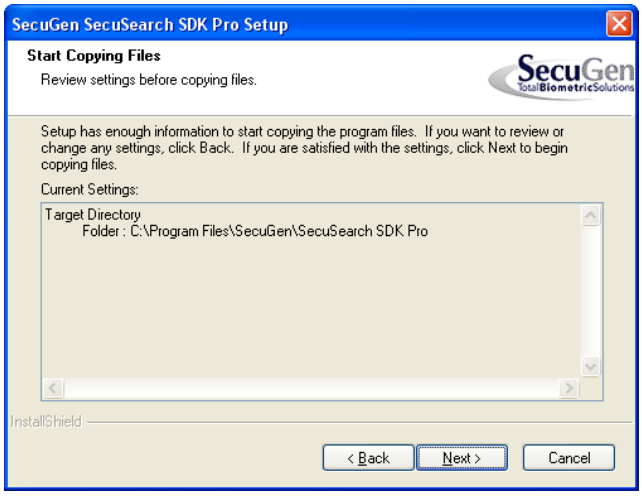
5. Choose the folder to copy to, and click **Next**.



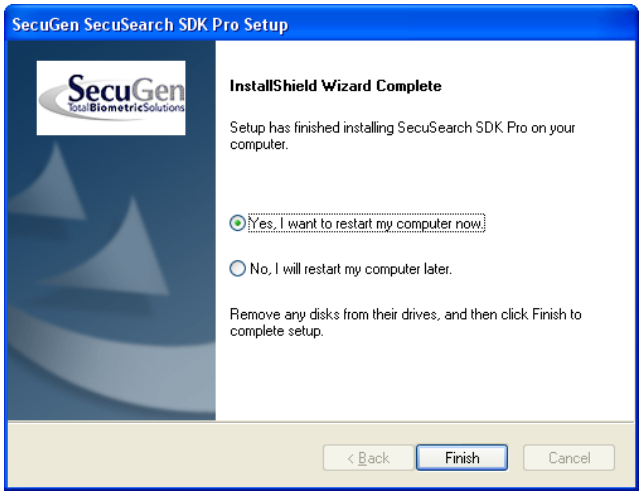
6. Read the SecuSearch License Policy carefully and check the option. A temporary license file will be created during installation. This temporary license will be valid for 30 days. Once expired, you have to contact your SecuGen representative to issue a new license file to reactivate.



7. To start copying files, click **Next**.

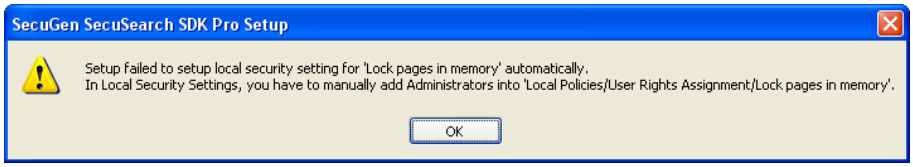


8. To complete installation, you must restart your system.

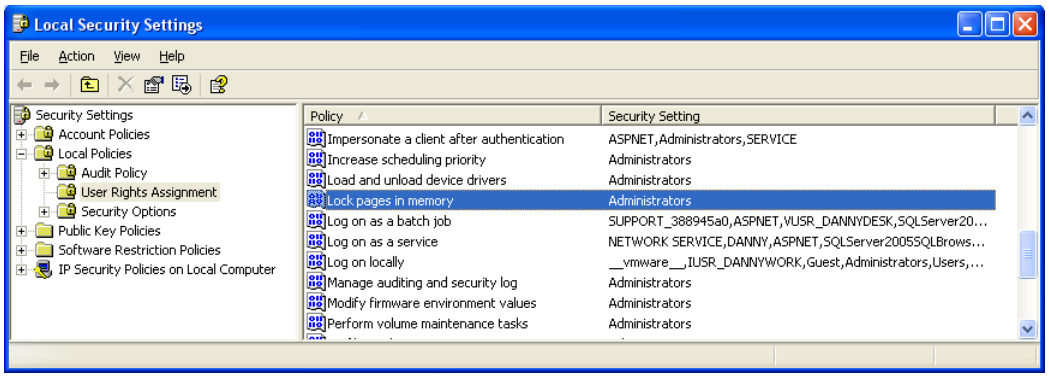


Manual Configuration

The SecuSearch SDK *Pro* setup automatically configures the local security setting that is required to run the SecuSearch engine. However, if setup fails for any reason, you may need to configure it manually.



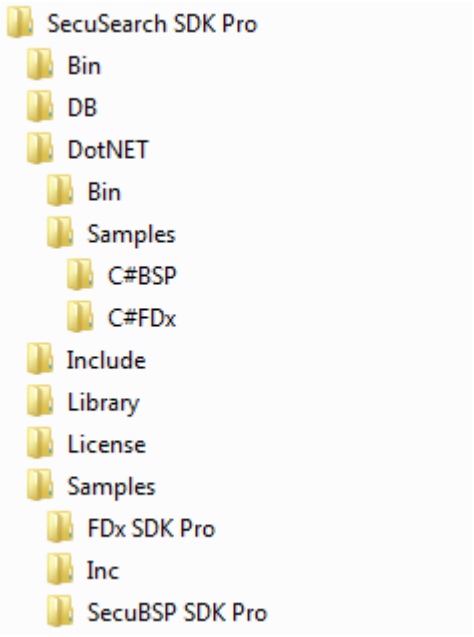
1. Run “Local Security Policy” from Control Panel / Administrative Tools.
2. Look into the tree name “User Rights Assignment” under Security Settings / Local Policies.
3. Select “Lock pages in memory” and open its Properties. Refer to the picture below.
   Add the “Administrators” group or specific users by selecting “Add User of Group…”. Only users with this privilege can start the SecuSearch engine. After updating this setting, restart the system to apply the changes.



IMPORTANT FOR WEB ENVIRONMENT:
For web environment, running the SecuSearch engine in IIS, you should specify additional user or group settings for “Lock pages in memory” as shown above. Also, for Windows 7, the IIS_IUSRS group is required; for Windows 2003, IIS_WPG is required.

2.3. Files to be installed

The directory structure below appears after the SecuSearch Engine installation is complete. Files will be copied according to this structure.



Note: During installation, a temporary license file, “temp_license.dat”, is created under the directory C:\Program Files\SecuGen\SecuSearch SDK Pro\License. To obtain a permanent SecuSearch license file from SecuGen, refer to the document, *SecuSearch License File Instructions*.

1. **\DB: database files**
   The database files are only for sample applications. You will need to have your own database for your own applications.
   SSTest.mdb – Saves user information and fingerprint data

2. **\DLL: Dll files**
   SecuSearchMx.dll – The SecuSearch Engine DLL

SecuSearchAPIMx.dll – The SecuSearch API DLL

3. **\Include: Header file**
   SecuSearchAPI.h – Defines SecuSearch interface APIs in the SecuSearch SDK
   SecuSearchDef.h – Defines the basic data type used in the SecuSearch Engine and error values (included in SecuSearchAPI.h)
   SecuSearchErr.h – Defines error codes returned by SecuSearch interface APIs

4. **\Library: Library file**
   SecuSearchAPIMx.lib – SecuSearch Interface API library file

5. **\License: Temporary license file**
   Temp_license.dat – Temporary SecuSearch license file (valid for 30 days and 20,000 templates)

6. **Samples\SecuBSP SDK Pro: Sample program (Visual C++)**
   \Bin – Contains the executable file of the sample program
   \Source – Contains the source code and project file for the SecuBSP *Pro* sample program

7. **Samples\FDx SDK Pro: Sample program (Visual C++)**
   \Bin – Contains the executable file of the sample program
   \Source – Contains the source code and project file for the FDx SDK *Pro* sample program

8. **DotNET\Bin: SecuSearch .NET Assembly file**
   SecuSearchMx.NET.DLL – SecuSearch .NET Component

9. **DotNET\Samples\C#BSP: Sample program in C# using SecuBSP SDK Pro**
   Sample program using SecuSearchMx.NET.DLL and SecuBSP SDK Pro

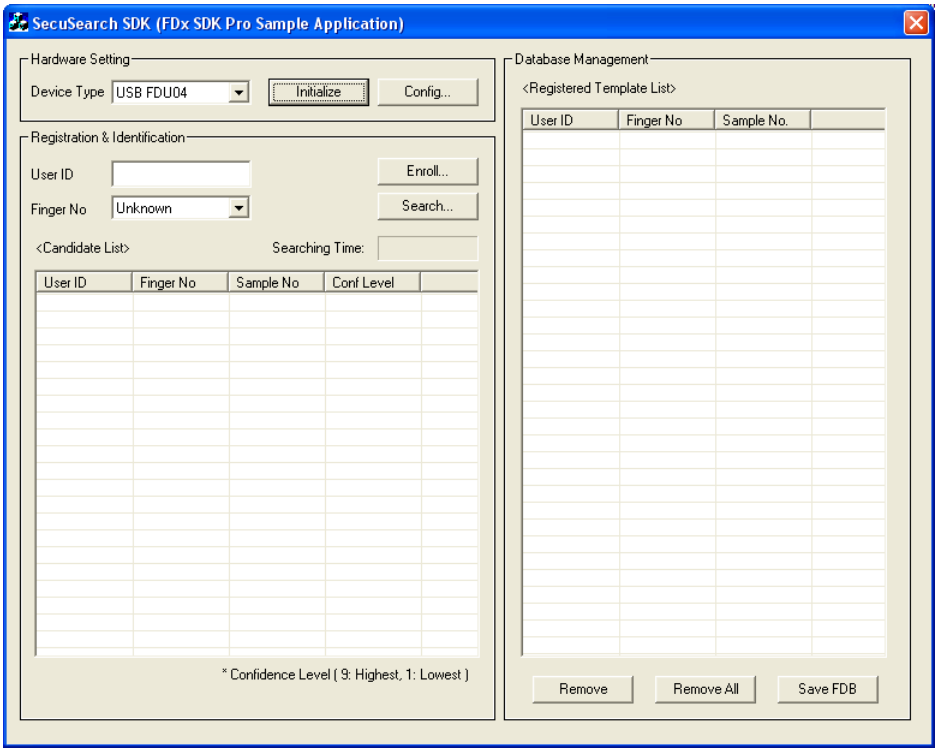
10. **DotNET\Samples\C#FDx: Sample program in C# using FDx SDK Pro**
    Sample program using SecuSearchMx.NET.DLL and FDx SDK Pro

Chapter 3. Using Sample Programs

3.1. FDx SDK Pro Sample Program

The 'FDX SDK *Pro*' sample program is a demo program for managing users. The FDx SDK *Pro* sample program captures fingerprints and extracts the minutiae data to perform registration and input functions. The FDx SDK *Pro* should be installed first before running the sample program.

Run the program '**VC++ Sample Using FDx SDK Pro**' from **Start> Program > SecuSearch SDK Pro > Samples**.



- **Hardware Settings**
  Fingerprint registration and searches can be performed using the reader only after it is initialized. Click on **Initialize** to initialize the SecuGen fingerprint reader and adjust its settings. Device initialization does not need to be done when removing or clearing database records. Click **Config…** to change the device settings after initialization.

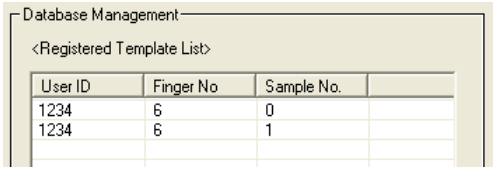
- **Registration**
  The registration process is for user fingerprint registration, which involves inputting user information and creating the fingerprint template. A user can register additional fingers in separate files, but two fingerprint samples from the same finger will be required per registration.

  1. Enter the user ID, choose the fingerprint to register, and click **Enroll…**.

  2. Two fingerprint samples from the same finger are needed for registration. Place a finger on the reader and click in the first **Capture** button (on the left). After the fingerprint image appears, release your finger

from the reader, and then place the same finger back on the sensor to repeat the process. Next, click in the second **Capture** button (on the right). The second fingerprint image will appear. Click **OK** to finish registration.

3. After the registration process is complete, the user ID, finger number and sample number can be viewed in the Database Management list. The sample number of the first registered fingerprint is '0' and the sample number of the second fingerprint is '1.'



- **Identification**
  The identification process takes an input (query) fingerprint and searches for the corresponding registered user in the database.

1. To begin, click **Search….**

2. From the 'Identification' dialog box, place a finger on the sensor and click in the **Capture** button to input a fingerprint. After the fingerprint appears, click **OK** to finish.

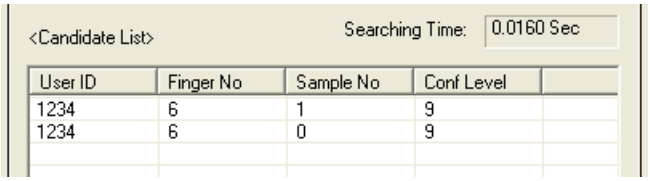
3. After the fingerprint is input, the SecuSearch Engine searches for the corresponding fingerprint template in the database, and displays the results.

   The results returned include the Search time and the Candidates List which consist of the following information for each candidate:

   User ID
   Finger No
   Sample No (e.g., 0 for the first fingerprint captured during registration)
   Confidence level

   The Confidence level is determined by the degree of similarity between the input fingerprint and the

candidates. Higher values mean that there is a higher possibility that both fingerprints are the same fingerprint. The confidence level ranges from 1 to 9.

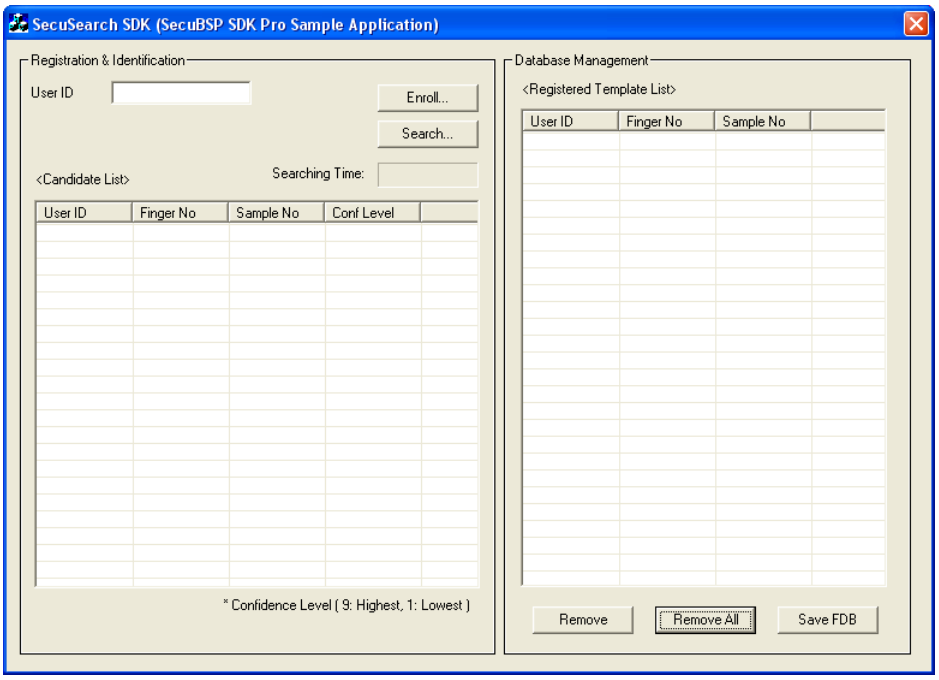


- **Database Management**
  To delete a record, select the record and click **Remove**. To delete all registered fingerprints at once, click **Remove All**. To save the SecuSearch data residing in the RAM into FDB file, click **Save FDB**.

3.2. SecuBSP® SDK Pro Sample Program

The 'SecuBSP SDK *Pro*' sample program is a program for managing users. The SecuBSP SDK *Pro* sample program captures fingerprints and extracts the minutiae data (FIR format) to perform fingerprint registration and input functions. The SecuBSP SDK *Pro* should be installed first before running the sample program.

Run the program '**VC++ Sample Using SecuBSP SDK Pro**' from **Start > Program >SecuSearch SDK Pro > Samples**.



- **Registration**
  The registration process is for user fingerprint registration, which involves inputting user information and creating the fingerprint template. A user can register additional fingers in separate files, but two fingerprint samples from the same finger will be required per registration.

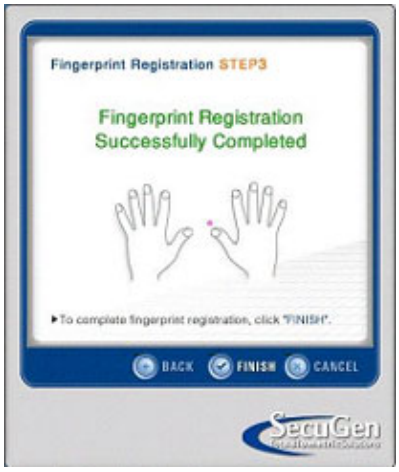
  1. Enter the user ID, choose the fingerprint to register, and click **Enroll…**.

  2. The SecuBSP registration dialog box will open as shown below. Click **NEXT** to proceed to the next step.

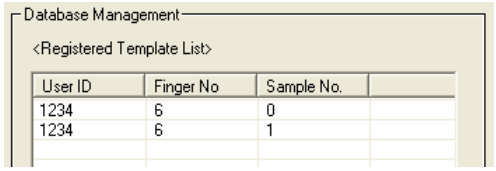
  3. Select the finger to be registered by clicking on the small blue circle above the fingertip.

  4. The following fingerprint capture dialog box is opened. Two fingerprint samples from the same finger will need to be captured to complete the registration process. Place one finger on the reader, and then release the finger when prompted. Place the finger a second time. If the fingerprint image is too bright or dark (i.e. poor quality), fingerprint registration may fail. In this case, click **ADJUST** to manually adjust the contrast setting for the fingerprint sensor.

5. When the final fingerprint capture is finished, click **NEXT**. If you want to register another finger, or re-register the same finger, click **BACK**. When done, click **FINISH**.



6. After registration is completed, the user name, finger number and sample number can be found in the “Database management” sheet as shown below.



- **Identification**
  The identification process takes an input (query) fingerprint and searches for the corresponding registered user in the database.

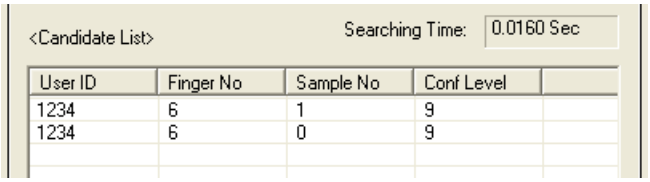
  1. To begin, click **Search….**

  2. When the fingerprint input dialog box appears, place a finger to be searched on the fingerprint reader.

  3. After the fingerprint is input, the SecuSearch Engine searches for the corresponding fingerprint template in the database, and displays the results.

     The results returned include the Search time and the Candidates List which consist of the following information for each candidate:
     User ID
     Finger No
     Sample No (e.g., 2 for the second fingerprint captured during registration)
     Confidence Level

  The Confidence level is determined by the degree of similarity between the input fingerprint and the candidates. Higher values mean that there is a higher possibility that both fingerprints are the same fingerprint. The confidence level ranges from 1 to 9.



- **Database Management**
  To delete a record, select the record and click **Remove**. To delete all registered fingerprints at once, click **Remove All**. To save the SecuSearch data residing in the RAM into a FDB file, click **Save FDB**.

Chapter 4. SDK Programming

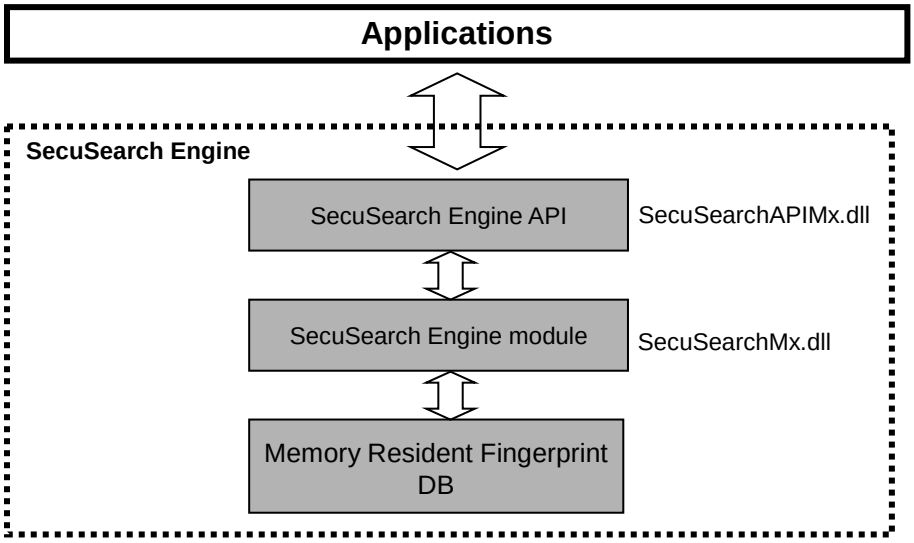
This chapter describes how to program with the one-to-many (1:N) matching system using the SecuSearch SDK.

4.1. SecuSearch Engine Structure

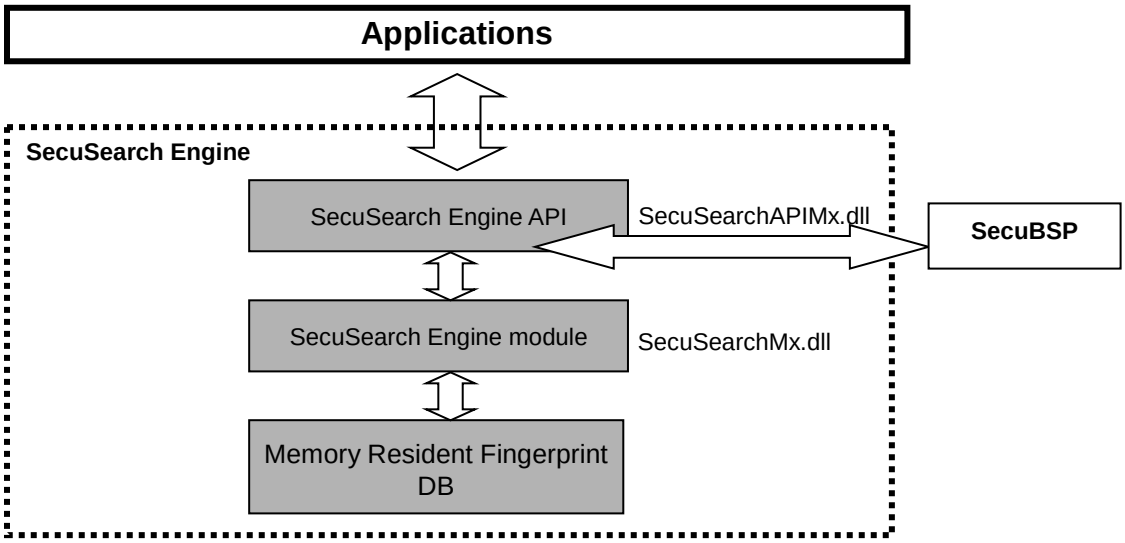
The SecuSearch Engine is able to use two types of fingerprint minutiae template formats as inputs:

- **The 400 byte template format** (as used by FDx SDK *Pro* and SDA0x / FDA0x DK)
- **The FIR template format** (as used by SecuBSP SDK *Pro*)

SecuSearch Engine Structure (400 byte template format)



SecuSearch Engine Structure (FIR template format)



Module description

- **SecuSearch Engine API**
  Interface between application program and SecuSearch Engine
- **SecuSearch Engine module**
  Main module that performs registration, deletion, search of fingerprint and internal DB management
- **Memory Resident Fingerprint DB**
  Internal memory resident DB that contains registered fingerprint information. The input templates are reprocessed with additional information to facilitate fast searching. Fingerprint information is stored in a hash table-like memory resident DB.
- **SecuBSP**
  SecuGen's Secure Biometric Service Provider provides fingerprint input and recognition functions

4.2. System composition

In this section, block diagrams are shown for each 1:N matching system used with SecuGen's fingerprint recognition products.

SecuGen fingerprint recognition products that work with the SecuSearch Engine:

FDx SDK *Pro*: SecuGen's low-level fingerprint recognition SDK that is used with SecuGen peripherals
SecuBSP SDK *Pro*: SecuGen's high-level fingerprint recognition SDK that is used with SecuGen peripherals
SDA0x / FDA0x DK: SecuGen's DK designed for SecuGen's built-in CPU type fingerprint recognition reader

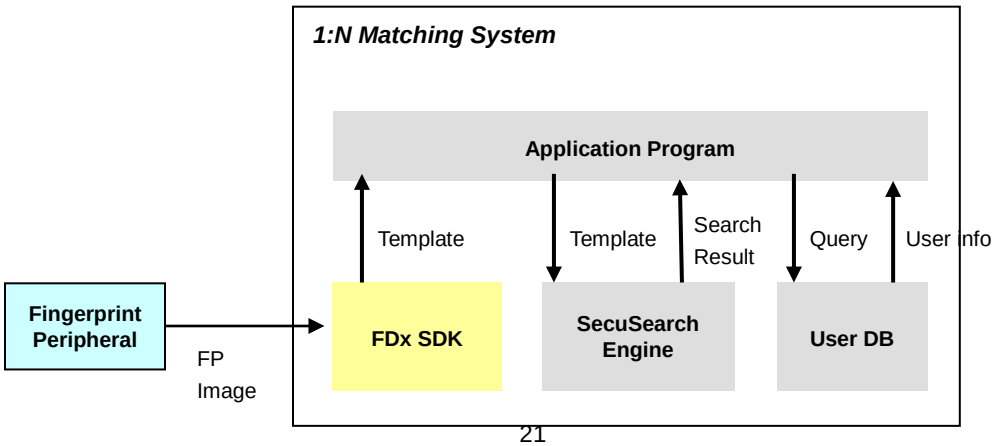
Note: In the system block diagrams below, the User DB is a database that contains registered user information. The User DB is different from the internal memory resident DB of the SecuSearch Engine.

There are two kinds of 1:N matching system:

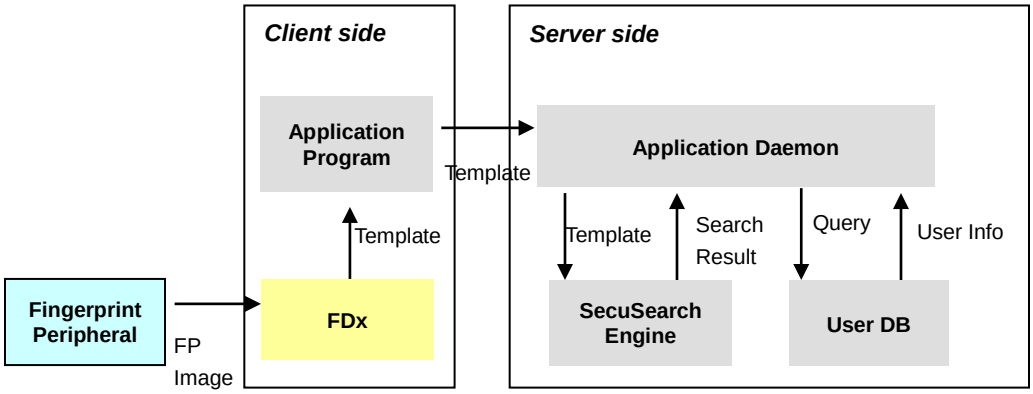
Stand-Alone type This 1:N matching system has a fingerprint reader, its device driver, a fingerprint feature (minutiae) extraction module, and the SecuSearch Engine, all in one system. The application program controls each module.

Client-Server type This 1:N matching system has two separated systems. The client system performs fingerprint capture and fingerprint feature (minutiae) extraction functions, and it transmits the resulting fingerprint template to the server. The server transmits the search results to the client after it performs 1:N matching.

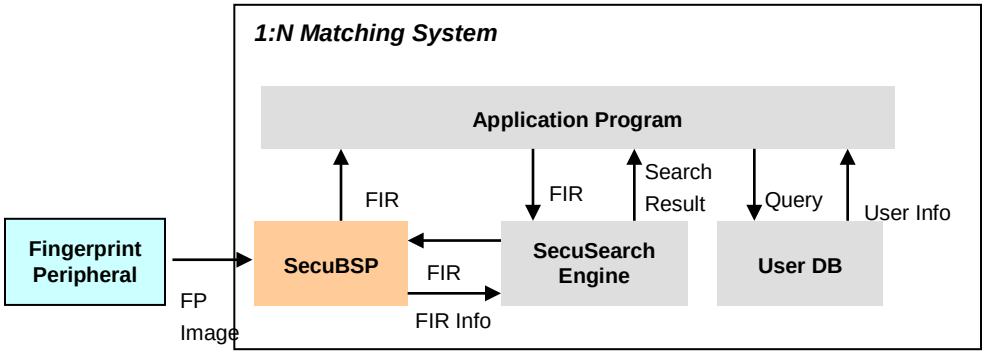
Stand-Alone type 1:N matching system with FDx SDK Pro



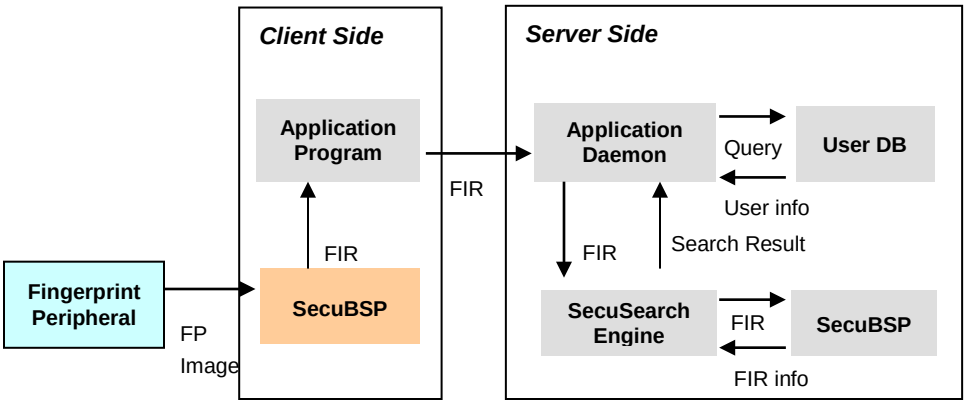
Client/Server type 1:N matching system with FDx SDK Pro



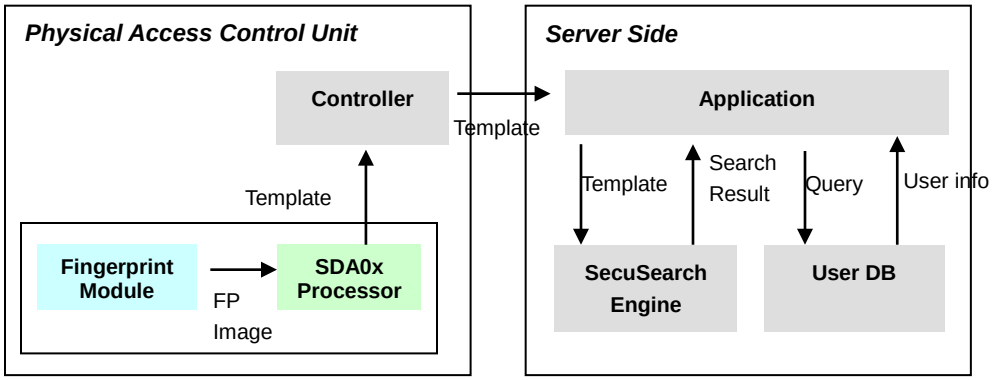
Stand-Alone type 1:N matching system with SecuBSP SDK Pro



Client/Server type 1:N matching system with SecuBSP SDK Pro



Client/Server type 1:N matching system with SDA0x or FDA0x



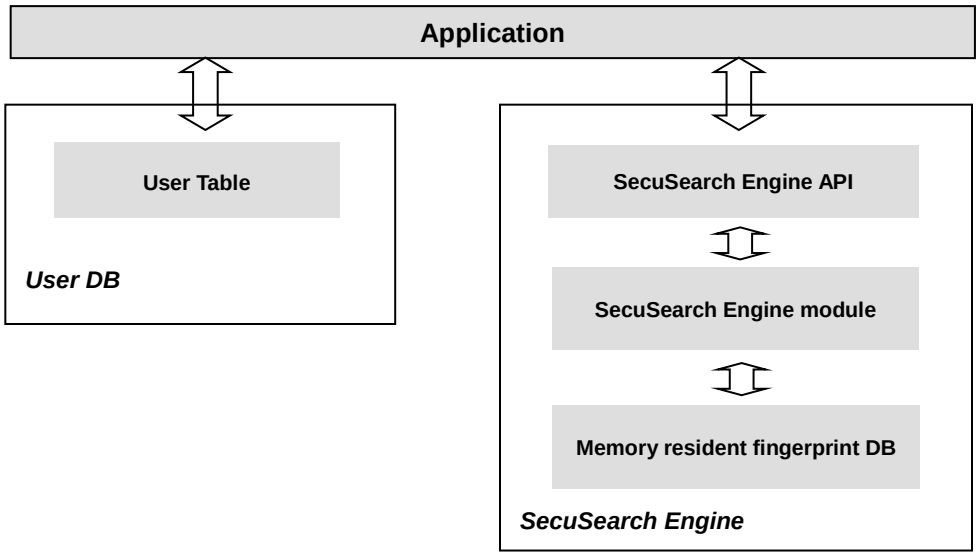
4.3. Recommended DB Management

The SecuSearch engine uses the database used in the 1:N fingerprint matching system:

- **Memory Resident Fingerprint DB**: An internal hash table-type fingerprint database that contains reprocessed fingerprint templates and additional information used to facilitate fast searching. This memory resident fingerprint DB exists inside of and is managed by the SecuSearch Engine.

- **User DB**: A database that contains information about the registered user, the fingerprint and the fingerprint minutiae template. It is managed by the application program and should be constructed by the application programmer. It can be made using any kind of commercial database (SQL, Oracle, etc). Sample applications included in the SDK use the MDB file with ODBC connection.

1:N Matching System Database Structure

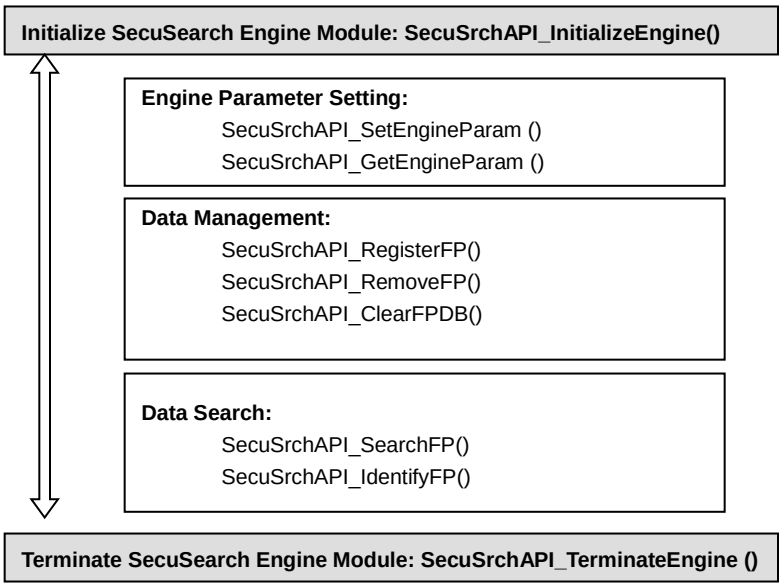


4.4. SecuSearch Engine APIs

The SecuSearch Engine APIs perform the following functions:

- Initialization/termination
- Engine parameter setting
- Main functions (fingerprint registration, deletion, search)
- Internal fingerprint DB management functions

SecuSearch Engine API Structure



4.4.1. SecuSearch Engine initialization/ termination/ parameter setting

To create a 1:N matching program using the SecuSearch Engine Module, first initialize the Engine using **SecuSrchAPI_InitializeEngine().** This API initializes engine parameters, initializes the internal global variables, allocates the memory of the fingerprint database residing in the RAM, and initializes it. Be sure to use **SecuSrchAPI_TerminateEngine()** to complete the SecuSearch Engine process and free up the memory of the SecuSearch Engine.

1. **SecuSearch Engine initialization**

```
DLLEXPORT DWORD WINAPI
SecuSrchAPI_InitializeEngine(ENGINE_PARAM_* param);
```

Allocates and initializes memory
INPUT: Parameters of search engine
OUTPUT: Error code

Initialize the SecuSearch Engine using **SecuSrchAPI_InitializeEngine().** When this initialization function is called, SecuSearch Engine opens the license file to check its validity in the directory specified by the SecuSearch Engine installer. If the license file does not exist or it is not valid, initialization will fail. Since the maximum number of fingerprints that can be registered is limited by the amount licensed, users cannot register their fingerprints more times than the maximum number allowed.

Engine parameters at the initialization stage are defined as follows:

```
typedef struct ENGINE_PARAM{
        CHAR  szLicenseFile[MAX_PATH];
        DWORD CandidateNumber;
        DWORD MemPoolSizeMB;
        DWORD reserved1;
        DWORD reserved2;
        DWORD reserved3;
        DWORD reserved4;
        DWORD reserved5;
        DWORD * reserved6;
        } ENGINE_PARAM_;
```

The szLicenseFile can be used to specify the SecuSearch license file issued by SecuGen Corp to activate the SecuSearch engine. The full path on the local drive should be used.

The CandidateNumber designates the number of candidates that will be listed when inquiring through the SecuSearch Engine. For instance, if the CandidateNumber is set at '10', then 10 candidates will be listed. MaxFingerNumber means the maximum number of fingerprints that can be registered in the DB.

The MemPoolSizeMB is used to allocate a specified amount of continuous memory block to load or create a fingerprint database residing in the RAM. This option value determines the maximum number of templates that can be registered. For instance, MemPoolSizeMB=500 means using 500 MB for a fingerprint database residing in the RAM in which an average of 30,000 to 40,000 templates can be registered. Too small of a value may not allow enough space for template registration, but too large of a value may cause an error when using the InitializeEngine or RegisterFP functions depending on the memory status of the system. Please refer to section 2.1. System Requirements, for more information about required memory.

```
ENGINE_PARAM_ g_Param; // SecuSearch Engine parameter.

strcpy(g_Param.szLicenseFile, "c:\\SecuGen\\License.dat");
g_Param.CandidateNumber = 30;
g_Param.MemPoolSizeMB = 100;

// Initialize Engine
If (SecuSrchAPI_InitializeEngine (&g_Param) != FPS_ERROR_NONE )
{
        // Init module failed. Show error message.
}
        // Init success.
```

2. Closing the SecuSearch Engine

DLLEXPORT DWORD WINAPI SecuSrchAPI_TerminateEngine(void);

Sets free the occupied memory
OUTPUT: Error code

Close the SecuSearch Engine using **SecuSrchAPI_TerminateEngine().** Be sure to use this function to free the allocated memory in the SecuSearch Engine module.

```
ERROR_TYPE_ ret;
ret = SecuSrchAPI_TerminateEngine ();        // Terminate BSP module
```

3. **Getting the SecuSearch Engine parameters**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_GetEngineParam(ENGINE_PARAM_* param);
   Sets free the occupied memory
   INPUT: Empty structure of parameters
   OUTPUT: Error code; current engine parameters

   Use **SecuSrchAPI_GetEngineParam()** to get the Engine parameters after SecuSearch Engine initialization.

```
ENGINE_PARAM_ g_Param; // SecuSearch Engine parameter
ERROR_TYPE_ ret;

ret = SecuSrchAPI_GetEngineParam (&g_Param); // get Engine parameter
```

4. **Setting the SecuSearch Engine parameters**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_SetEngineParam(ENGINE_PARAM_* param);
   Sets Parameters of search engine
   INPUT: Parameters of search engine
   OUTPUT: Error code

   Use **SecuSrchAPI_SetEngineParam()** to set new Engine parameters after SecuSearch Engine initialization.

```
ENGINE_PARAM_ g_Param; // SecuSearch Engine parameter
ERROR_TYPE_ ret;

strcpy(g_Param.szLicenseFile, “c:\\SecuGen\\License.dat”);
g_Param.CandidateNumber = 30;
g_Param.MemPoolSizeMB = 100;

ret = SecuSrchAPI_SetEngineParam (&g_Param); // Set Engine parameter
```

4.4.2. Fingerprint registration/ deletion/ searching

Fingerprint registration, deletion, and search are the main functions of the SecuSearch Engine. When a fingerprint is registered, one of two kinds of SecuGen fingerprint minutiae template formats may be used. One is the **400 bytes template format** that is extracted by SecuGen's **FDx SDK** or **FDA0x DK**, and the other is the **FIR template format** that is extracted by SecuGen's **SecuBSP SDK**. The SecuSearch Engine provides two different APIs to support both template formats.

1. **Registering the fingerprint using FDx SDK or FDA0x DK (400 byte template format)**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_RegisterFP(BYTE *feat_buff, ID_INFO_* user_id);
   Registers fingerprint template
   INPUT: Minutiae 400 byte template; ID information
   OUTPUT: Error code

```
typedef struct ID_INFO{
        DWORD   ID;
        BYTE    FingerNumber;
        BYTE    SampleNumber;
} ID_INFO_;
```

The SecuSearch Engine does not manage user information; it only takes input fingerprint data as a separate fingerprint. The application program must provide user management functionality by including a database (*User DB*) for the management of user information. This is the responsibility of the application programmer, and any kind of database can be used. The database used by the VC_ODBC sample and FDx SDK sample demo programs were created using MS Access. (See also section 4.3 DB structure.)

When a new user registers a fingerprint, the record will be added to the *User table*. The serial number (user_id) will increase by 1, and the name of the user is entered into the user_name field. At this time, the fingerprint_id updates automatically. In addition, the information regarding which fingerprint is being used (finger_num) and the number of times a user has input the same fingerprint (Sample_num) should be saved to the fingerprint table.

Upon completion of the above steps, call *SecuSrchAPI_RegisterFP()* to register the fingerprint minutiae template extracted through SecuGen's *FDx SDK* or *FDA0x DK* to the database residing in the RAM of the SecuSearch Engine.

```
ERROR_TYPE_ ret;                    // error message
BYTE feat_buff[TEMPLATE_SIZE];      // fingerprint data template
ID_INFO_ user_id;                   // input fingerprint information

ret = SecuSrchAPI_RegisterFP( feat_buff, &user_id ); // register fingerprint
```

The user_id contains information about the input fingerprint. Set the user_id, finger_num, and sample_num at the same values as those in the fingerprint table of the user database.

2. Registering the fingerprint using SecuBSP SDK (FIR template format)

As described in the SecuBSP SDK manual, a FIR format fingerprint minutiae template extracted by SecuBSP may contain information about more than one fingerprint per user. In other words, if a user registers multiple fingers, all the fingerprint minutiae data and their information are encoded into a single FIR. On the other hand, since individual fingerprint information should be written into the User DB as described in section 4.3, the application programmer must obtain individual fingerprint information from the FIR and write it in the User DB one by one. To register a FIR format fingerprint template, call **SecuSrchAPI_GetFIRInfo()** first, and then write each fingerprint information to the User DB. Then, call ***SecuSrchAPI_RegisterFPEx()*** or ***SecuSrchAPI_RegisterFPExW()*** to register the FIR template.

- **Getting individual fingerprint information from a FIR template**

DLLEXPORT DWORD WINAPI
SecuSrchAPI_GetFIRInfo(const char* szFIR, FIR_INFO_* pFIRInfo);

Gets fingerprint information from FIR (FingerNumber, SampleNumber)
INPUT: FIR Minutiae template (character string)
OUTPUT: Total number of fingerprints; FingerNumber & SampleNumber for each finger in the FIR

DLLEXPORT DWORD WINAPI
SecuSrchAPI_GetFIRInfoW(BSTR bstrFIR, FIR_INFO_* pFIRInfo);

Gets fingerprints information from FIR (FingerNumber, SampleNumber)
INPUT: FIR Minutiae template (wide character)
OUTPUT: Total number of fingerprints; FingerNumber & SampleNumber for each finger in the FIR

```
typedef struct FP_INFO
{
    BYTE FingerNumber;
    BYTE SampleNumber;
} FP_INFO_;
```

```
typedef struct FIR_INFO
{
    BYTE FingerCount;
    FP_INFO_ FPInfo[20];
} FIR_INFO_;
```

As described above, a FIR format template extracted by the SecuBSP SDK may contain fingerprint information of more than one finger per user. The output of ***SecuSrchAPI_GetFIRInfo()*** is FIR_INFO_ which contains the total number of fingerprints in the FIR and fingerprint information of each finger (FPInfo). FP_INFO_ contains information about the finger number and sample number. Using FP_INFO_, the application programmer should add the record of the fingerprint information to be registered, before calling the SecuSearch Engine's registration API. Since SecuBSP restricts the maximum number of samples to 2, the maximum number of fingerprints that a FIR template can have is 20 (i.e. 10 fingers x 2 samples).

```
ERROR_TYPE_ ret;                // error message
const char* szFIR;              // FIR (fingerprint data template)
FIR_INFO_ pFIRInfo;             // FIR information

ret = SecuSrchAPI_GetFIRInfo(szFIR, &pFIRInfo);
```

- **Registering a FIR template**
  Register a FIR format fingerprint minutiae template using the following APIs:
  SecuSrchAPI_RegisterFPEx: Registration API for character string type FIR
  SecuSrchAPI_RegisterFPExW: Registration API for wide character type FIR

  DLLEXPORT DWORD WINAPI
  SecuSrchAPI_RegisterFPEx(const char* szFIR, ID_INFO_* user_id);
  Registers fingerprint template
  INPUT: FIR format fingerprint minutiae template (character string type); ID information
  OUTPUT: Error code

  DLLEXPORT DWORD WINAPI
  SecuSrchAPI_RegisterFPExW(BSTR bstrFIR, ID_INFO_* user_id);
  Registers fingerprint template
  INPUT: FIR format fingerprint minutiae template (wide character type); ID information
  OUTPUT: Error code

  Regarding ID information (ID_INFO_), since the FIR already contains FingerNumber and SampleNumber, the application programmer does not have to write them as an ID_INFO_ input, but rather simply input the ID which is referred from the User DB (user_id, see section 4.3).

```
ERROR_TYPE_ ret;                // error message
const char* szFIR;              // FIR (fingerprint data template)
ID_INFO_* user_id;              // FIR information

ret = SecuSrchAPI_ RegisterFPEx (szFIR, user_id);
```

3. **Deleting a fingerprint**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_RemoveFP(ID_INFO_* user_id);
   Removes one fingerprint template for the specified user/finger/sample from the database
   INPUT: ID information
   OUTPUT: Error code

To delete a registered fingerprint, get the ID_INFO_ of the fingerprint from the user database. Delete the fingerprint from the SecuSearch Engine database using ***SecuSrchAPI_RemoveFP().*** The records in the user database (User DB) managed by the application program should also be deleted along with the SecuSearch Engine database.

To delete all fingerprints from a specific user in the SecuSearch Engine database, collect all appropriate fingerprint information from the fingerprint table of the user database (User DB) and delete each fingerprint separately using ***SecuSrchAPI_RemoveFP().***

```
ERROR_TYPE_ ret;                    // error message
ID_INFO_ user_id;                   // fingerprint information to be removed

ret = SecuSrchAPI_RemoveFP( &user_id );
```

DLLEXPORT DWORD WINAPI
SecuSrchAPI_RemoveFPEx(ID_INFO_* user_id);
Removes all fingerprint templates for the specified user from the database
INPUT: ID information
OUTPUT: Error code

4. Searching for a fingerprint

a. Search (Get candidates of similar fingerprints)

DLLEXPORT DWORD
WINAPI SecuSrch_SearchFP(BYTE *feat_buff,
CAND_LIST_ *candidate,
DWORD* num_candidate);

Searches for matching fingerprint candidates in DB (400 byte template)
INPUT: 400 byte minutiae template (from FDx SDK or FDA0x DK); empty candidate list buffer; reserved (NULL)
OUTPUT: Candidate list; Number of candidates; Error code

```
typedef struct CAND_LIST{
    DWORD   ID;
    BYTE    FingerNumber;
    BYTE    SampleNumber;
    BYTE    ConfidenceLevel;
} CAND_LIST_;
```

Use ***SecuSrchAPI _SearchFP()*** to get fingerprint candidates that match the input (query) fingerprint. Input the fingerprint data template that was extracted by the SecuGen fingerprint minutiae extraction algorithm and the empty candidate list buffer to get the information of the closest matching fingerprint. The candidate list includes the user number (ID), finger type (FingerNumber), and confidence level (ConfidenceLevel). The candidates are listed in descending order of confidence level, where a higher number means greater similarity to the input fingerprint and a lower number means less similarity to the input (query) fingerprint. When the confidence level is higher than 5, treat the fingerprint as a match.

```
ERROR_TYPE_ ret;                        // error message
BYTE feat_buff[TEMPLATE_SIZE];          // input fingerprint data template
CAND_LIST_ candidate[CANDIDATE_NUM];    // output candidate list

ret = SecuSrchAPI_SearchFP( feat_buff, &candidate, NULL ); // search
fingerprint from DB
```

Search using FIR template

```
DLLEXPORT DWORD WINAPI
SecuSrchAPI_SearchFPEx(     const char* szFIR,
                            CAND_LIST_ *candidate,
                            DWORD* num_candidate );
```

Searches for matching fingerprint candidates in DB (FIR template)
INPUT: FIR format fingerprint minutiae template (char string type); empty candidate list buffer; reserved (NULL)
OUTPUT: Candidate list; Number of candidates; Error code

```
DLLEXPORT DWORD WINAPI
SecuSrchAPI_SearchFPExW(BSTR bstrFIR, CAND_LIST_ *candidate, void* );
```

Searches for matching fingerprint candidates in DB (FIR template)
INPUT: FIR format fingerprint minutiae template (wide char type); empty candidate list buffer; reserved (NULL)
OUTPUT: Candidate list; Error code

b. Identification

```
DLLEXPORT DWORD WINAPI
SecuSrchAPI_IdentifyFP(BYTE *feat_buff, unsigned char seculevel, ID_INFO_ *result)
```

Identifies matching fingerprint in DB based on a specified security level (400 byte template)
INPUT: 400 byte minutiae template; security level; search result
OUTPUT: ID_INFO_ structure buffer; error code

```
typedef struct ID_INFO{
    DWORD   ID;
    BYTE    FingerNumber;
    BYTE    SampleNumber;
} ID_INFO_;
```

Use ***SecuSrchAPI_IdentifyFP()*** to identify the input (query) fingerprint. Input the fingerprint data template (extracted by the SecuGen fingerprint minutiae extraction algorithm), the security level, and the empty ID_INFO structure. If there is a matching fingerprint, the fingerprint information will be printed in the ID_INFO structure; otherwise all information will be set to '0'. Fingerprint information includes user number (ID), finger type (FingerNumber), and number of times the same fingerprint has been input.

The security level is set to determine to what degree the fingerprints match. If the confidence level of the searched fingerprint is higher than the security level, the fingerprints are considered to be matching. If the confidence level is lower than the security level, the fingerprints are considered to be not matching.

```
ERROR_TYPE_ ret;                // error message
BYTE feat_buff[TEMPLATE_SIZE];  // input fingerprint data template
unsigned char security;         // security level
ID_INFO_ result;                // output fingerprint information

ret = SecuSrchAPI_IdentifyFP ( feat_buff, security, &result ); // identify the
fingerprint from DB
```

Identification using FIR template

```
DLLEXPORT DWORD WINAPI
SecuSrchAPI_IdentifyFPEx(const char* szFIR, unsigned char seculevel, ID_INFO_ *result);
```

Identifies matching fingerprint in DB based on a specified security level (FIR template)

INPUT: FIR format fingerprint minutiae template (char string); security level; ID_INFO_ structure buffer
OUTPUT: ID_INFO_ structure buffer; Error code

DLLEXPORT DWORD WINAPI
SecuSrchAPI_IdentifyFPExW(BSTR bstrFIR, unsigned char seculevel, ID_INFO_ *result);
Identifies matching fingerprint in DB based on a specified security level (FIR template)
INPUT: FIR format fingerprint minutiae template (wide char); security level; ID_INFO_ structure buffer
OUTPUT: ID_INFO_ structure buffer; Error code

4.4.3. Managing the Fingerprint DB

The fingerprint database of the SecuSearch Engine resides in the RAM and therefore must be saved to disk when a fingerprint is registered or deleted. Also, when rebooting the system, the database stored in the disk should be loaded to memory just after initializing the SecuSearch Engine.

1. **Clearing the fingerprint database**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_ClearFPDB();
   Deletes all fingerprint data in the fingerprint database residing in the RAM of the SecuSearch Engine
   INPUT: None
   OUTPUT: Error code

```
ERROR_TYPE_ ret;                            // error message

ret = SecuSrchAPI_ClearFPDB();              // Clear all data on search DB
```

2. **Reading the count of registered fingerprints**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_GetFPCount(DWORD* count);
   Reads the count of registered fingerprints in the fingerprint database residing in the RAM of the SecuSearch Engine
   INPUT: None
   OUTPUT: Data count; Error code

```
ERROR_TYPE_ ret;                            // error message
DWORD count = 0;

// Read data count on search DB
ret = SecuSrchAPI_GetFPCount(&count);       // Read data count on search DB
```

3. **Reading the list of registered fingerprints**
   DLLEXPORT DWORD WINAPI
   SecuSrchAPI_GetFPList(DWORD buff_size, ID_INFO_ idList, DWORD* count);
   Reads the list of ID_INFO_ of registered fingerprints in the fingerprint database residing in the RAM of the SecuSearch Engine
   INPUT: Size of ID_INFO_ structure
   OUTPUT: ID_INFO_; Data count; Error code

```
ERROR_TYPE_ ret;                            // error message
DWORD buff_size = 20000, count = 0;
ID_INFO_* idList = NULL;
idList = new ID_INFO_[buff_size];
```

```
//Read data list on search DB
ret = SecuSrchAPI_GetFPList(buff_size, idList, &count);
delete[] idList;
```

4. **Reading the template data**

DLLEXPORT DWORD WINAPI
SecuSrchAPI_GetFPData(ID_INFO_* idInfo,BYTE* feat_buff);

Reads the template data in the fingerprint database residing in the RAM of the SecuSearch Engine
INPUT: ID_INFO_ structure
OUTPUT: Template data, Error code

```
ERROR_TYPE_ ret;                              // error message
BYTE feat_buff[400];
ID_INFO_ idInfo;
idInfo.ID = 1234;
idInfo.FingerNumber = 1;
idInfo.SampleNumber = 1;

// Read template data on search DB
ret = SecuSrchAPI_GetFPData(&idInfo, feat_buff);
```

Chapter 5. SDK .NET Programming

The SecuSearch .NET library (SecuSearch.NET.DLL) is designed for .NET application developers to easily use SecuSearch SDK functionalities in the .NET environment. The SecuSearch .NET component is a wrapper class of SecuSearchAPI.DLL. Most of the SDK functions are implemented in the SecuSearch class in SecuSearch.NET.DLL. Therefore all functions provided in SecuSearch.NET.DLL are the same as those provided in SecuSearchAPI.DLL.

This chapter explains how to use the SecuSearch class in your .NET application. Since the SecuSearch.NET library uses the SecuSearch SDK, SecuSearch SDK runtime files (SecuSearch.DLL and SecuSearchAPI.DLL) should be in the execution path. The sample program in SecuSearch .NET requires the FDx SDK or SecuBSP SDK to get template data from the SecuGen reader.

- **Software Requirements:**
  - .NET Framework 2.0
  - SecuSearch SDK Pro

- **Software Requirements for sample program**
  - .NET Framework 2.0
  - Visual Studio 2005 or higher
  - SecuSearch SDK Pro
  - FDx SDK Pro or SecuBSP SDK Pro

5.1. Include SecuSearch.NET.DLL

To use the SecuSearch.NET library (SecuSearch class), you should include the SecuSearch.NET.DLL file as a reference. (Right-click on the Project File, and then select '**Add references…**'. From the .NET tab, click '**Browse**' and locate the file SecuSearch.NET.DLL). The SecuSearch.NET.DLL file is located in the Bin directory of the installation directory.

5.2. Namespace

The SecuSearch class is in the SecuGen.SecuSearchSDK namespace. Specify the SecuGen.SecuSearchSDK namespace in the source file where the SecuSearch class is used.

```
[C#]
using  SecuGen.SecuSearchSDK;

[VB.NET]
imports SecuGen.SecuSearchSDK
```

5.3. Programming

5.3.1 Create SecuSearch Class

To use the **SecuSearch** class, the **SecuSearch** class must first be instantiated. This is done by calling the **SecuSearch ()** class constructor.

```
[C#]
private SecuSearch m_SecuSearch;
…
m_SecuSearch = new SecuSearch();

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

5.3.2. Initialize SecuSearch Engine

Initialize the SecuSearch Engine using **InitializeEngine().** To use the SecuSearch engine, it should be called.

```
[C#]
pEngineParam = new SS_EngineParam();
pEngineParam.szLicenseFile = “c:\\SecuGen\\License.dat”;
pEngineParam.CandidateNumber = 30;
pEngineParam.MemPoolSizeMB = 500;
iError = m_SecuSearch.InitializeEngine(pEngineParam);

[VB.NET]
Dim iError As Int32
Dim pEngineParam As SS_EngineParam

pEngineParam = New SS_EngineParam
pEngineParam.szLicenseFile = “c:\\SecuGen\\License.dat”
pEngineParam.CandidateNumber = 30
pEngineParam.MemPoolSizeMB = 500

iError = m_SecuSearch.InitializeEngine(pEngineParam)
```

5.3.3. Close SecuSearch Engine

Close the SecuSearch Engine using **TerminateEngine().** Before terminating the SecuSearch engine, it should be called.

```
[C#]
m_SecuSearch.TerminateEngine();

[VB.NET]
m_SecuSearch.TerminateEngine()
```

5.3.4. Set Engine Parameter

Change the current SecuSearch engine parameter using **SetEngineParam()**

```
[C#]
Int32 iError;
```

```
SS_EngineParam pEngineParam;

pEngineParam = new SS_EngineParam();
pEngineParam.szLicenseFile = “c:\\SecuGen\\License.dat”;
pEngineParam.CandidateNumber = 30;
pEngineParam.MemPoolSizeMB = 500;
iError = m_SecuSearch.SetEngineParam(pEngineParam);

[VB.NET]
Dim pEngineParam As SS_EngineParam
Dim iError As Int32

pEngineParam = New SS_EngineParam
pEngineParam.szLicenseFile = “c:\\SecuGen\\License.dat”
pEngineParam.CandidateNumber = 30
pEngineParam.MemPoolSizeMB = 500
iError = m_SecuSearch.SetEngineParam(pEngineParam)
```

5.3.5. Get Engine Parameter

Get the current SecuSearch engine parameter using **GetEngineParam()**

```
[C#]
Int32 iError;
SS_EngineParam pEngineParam;

pEngineParam = new SS_EngineParam();
iError = m_SecuSearch.GeEngineParam(pEngineParam);

[VB.NET]
Dim pEngineParam As SS_EngineParam
Dim iError As Int32

pEngineParam = New SS_EngineParam
iError = m_SecuSearch.GetEngineParam(pEngineParam)
```

5.3.6. Register Fingerprint

Add fingerprint data to the SecuSearch Engine database. Depending on which template format is used, use **RegisterFP()** or **RegisterFPEx().**

- **For FDx SDK (400 byte template format)**
  Use **RegisterFP()**

```
[C#]
Int32 iError;
Byte[] fp_image;
Byte[] minData;
SS_IDInfo idInfo;
String idInfoStr;

minData = new Byte[400];
```

```
//Get 400 byte template using FDx SDK
fp_image = new Byte[m_ImageWidth * m_ImageHeight];

iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50);
if (iError != 0)
{
   StatusBar.Text = "Image Capture Error: " + Convert.ToString(iError);
   return;
}

// Get minData
iError = m_FPM.GetMinutiae(fp_image, minData);
if (iError != 0)
{
   StatusBar.Text = "Get Minutiae Error: " + Convert.ToString(iError);
   return;
}

// Add minData to SecuSearch Database
idInfo = new SS_IDInfo();
idInfo.ID = Convert.ToInt32(IdText.Text);
idInfo.FingerNumber = (Byte)FNComboBox.SelectedIndex;   //Finger number
idInfo.SampleNumber = Convert.ToByte(SNText.Text);      //Sample number

iError = m_SecuSearch.RegisterFP(minData, idInfo);
```

[VB.NET]

```
Dim iError As Int32
Dim fp_image() As Byte
Dim minData(400) As Byte
Dim idInfo As SS_IDInfo

'Get 400 byte template using FDx SDK
ReDim fp_image(m_ImageWidth * m_ImageHeight)
iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50)
iError = m_FPM.GetMinutiae(fp_image, minData)

'Add minData to SecuSearch Database
idInfo = New SS_IDInfo
idInfo.ID = Convert.ToInt32(IdText.Text)
idInfo.FingerNumber = FNComboBox.SelectedIndex  'Finger number
idInfo.SampleNumber = Convert.ToByte(SNText.Text) 'Sample number

iError = m_SecuSearch.RegisterFP(minData, idInfo)
```

- **For SecuBSP SDK (FIR template format)**
  Use **RegisterFPEx()**

[C#]

```
Int32 iError;
String firData;
SS_IDInfo idInfo;
SS_FIRInfo firInfo;
String firInfoStr;
```

```
iError = (Int32)m_SecuBSP.Enroll("");
if (iError == (Int32)BSPError.ERROR_NONE)
   firData = m_SecuBSP.FIRTextData;
else
   return;

// Add firText to SecuSearch Database
idInfo = new SS_IDInfo();
idInfo.ID = Convert.ToInt32(IdText2.Text);

// firText has already FingerNumber and Sample number info inside.
// so don't need to set FingerNumber and SampleNumber
iError = m_SecuSearch.RegisterFPEx(firData, idInfo);
```

[VB.NET]

```
Dim iError As Int32
Dim firData As String
Dim idInfo As SS_IDInfo
Dim firInfo As SS_FIRInfo

iError = m_SecuBSP.Enroll("")
If (iError = BSPError.ERROR_NONE) Then
   firData = m_SecuBSP.FIRTextData
End If

'Add firText to SecuSearch Database
idInfo = New SS_IDInfo
idInfo.ID = Convert.ToInt32(IdText2.Text)

' firText has already FingerNumber and Sample number info inside.
' so don't need to set FingerNumber and SampleNumber
iError = m_SecuSearch.RegisterFPEx(firData, idInfo)
```

5.3.7. Search Fingerprints

Search for matching fingerprint candidates in the SecuSearch Engine database. Depending on which template format is used, use **SearchFP()** or **SearchFPEx()**.

- **For FDx SDK (400 byte template format)**
  Use **SearchFP()**

[C#]

```
Int32 iError;
Byte[] fp_image;
Byte[] minData;
SS_CandList[] candArray;
String candInfoStr;
Int32 count_candidate = 0;

minData = new Byte[400];
fp_image = new Byte[m_ImageWidth * m_ImageHeight];
candArray = new SS_CandList[m_NumCandList];
```

```
iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50);
iError = m_FPM.GetMinutiae(fp_image, minData);

iError = m_SecuSearch.SearchFP(minData, candArray, ref count_candidate);
```

[VB.NET]

```
Dim iError As Int32
Dim fp_image() As Byte
Dim minData(400) As Byte
Dim candArray() As SS_CandList
Dim count_candidate As Int32

ReDim fp_image(m_ImageWidth * m_ImageHeight)
iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50)
iError = m_FPM.GetMinutiae(fp_image, minData)

ReDim candArray(m_NumCandList)

iError = m_SecuSearch.SearchFP(minData, candArray, count_candidate)
```

- **For SecuBSP SDK (FIR template format)**
  Use **SearchFPEx()**

[C#]

```
Int32 iError;
String firData;
SS_CandList[] candArray;
String candInfoStr;
Int32 count_candidate = 0;

candArray = new SS_CandList[m_NumCandList];

iError = (Int32)m_SecuBSP.Capture(FIRPurpose.VERIFY);

if (iError != (Int32)BSPError.ERROR_NONE)
{
   StatusBar.Text = "SecuBSP Capture() Error: " + Convert.ToString((Int32)iError);
   return;
}

firData = m_SecuBSP.FIRTextData;
iError = m_SecuSearch.SearchFPEx(firData, candArray, ref count_candidate);
```

[VB.NET]

```
Dim iError As Int32
Dim firData As String
Dim candArray() As SS_CandList
Dim i As Int32
Dim candInfoStr As String
Dim count_candidate As Int32

iError = m_SecuBSP.Capture(FIRPurpose.VERIFY)
If (iError = BSPError.ERROR_NONE) Then
   firData = m_SecuBSP.FIRTextData
```

```
End If

ReDim candArray(m_NumCandList)

iError = m_SecuSearch.SearchFPEx(firData, candArray, count_candidate)
```

5.3.8. Identify Fingerprint

Identify a matching fingerprint in the SecuSearch engine database. Depending on which template format is used, use **IdentifyFP()** or **IdentifyFPEx().**

- **For FDx SDK (400 byte template format)**
  Use **IdentifyFP()**

```
[C#]
Int32 iError;
Byte[] fp_image;
Byte[] minData;
SS_IDInfo idInfo;
String idInfoStr;

minData = new Byte[400];
fp_image = new Byte[m_ImageWidth * m_ImageHeight];

iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50);
iError = m_FPM.GetMinutiae(fp_image, minData);

idInfo = new SS_IDInfo();
iError = m_SecuSearch.IdentifyFP(minData, 5, idInfo);

[VB.NET]
Dim iError As Int32
Dim fp_image() As Byte
Dim minData(400) As Byte
Dim idInfo As SS_IDInfo
Dim idInfoStr As String

ReDim fp_image(m_ImageWidth * m_ImageHeight)
iError = m_FPM.GetLiveImageEx(fp_image, 5000, PictureBox1.Handle.ToInt32(), 50)
iError = m_FPM.GetMinutiae(fp_image, minData)

idInfo = New SS_IDInfo
iError = m_SecuSearch.IdentifyFP(minData, 5, idInfo)
```

- **For SecuBSP SDK (FIR template format)**
  Use **IdentifyFPEx()**

```
[C#]
Int32       iError;
String      firData;
SS_IDInfo   idInfo;
String      idInfoStr;
```

```
iError = (Int32)m_SecuBSP.Capture(FIRPurpose.VERIFY);

if (iError != (Int32)BSPError.ERROR_NONE)
{
   StatusBar.Text = "SecuBSP Capture() Error: " + Convert.ToString((Int32)iError);
   return;
}

firData = m_SecuBSP.FIRTextData;
idInfo = new SS_IDInfo();
iError = m_SecuSearch.IdentifyFPEx(firData, 5, idInfo);
```

[VB.NET]

```
Dim iError As Int32
Dim fp_image() As Byte
Dim firData As String
Dim idInfo As SS_IDInfo
Dim idInfoStr As String

iError = m_SecuBSP.Capture(FIRPurpose.VERIFY)
If (iError = BSPError.ERROR_NONE) Then
   firData = m_SecuBSP.FIRTextData
End If

idInfo = New SS_IDInfo
iError = m_SecuSearch.IdentifyFPEx(firData, 5, idInfo)
```

5.3.9. Delete Fingerprint

To delete one template from the SecuSearch Engine database, use **RemoveFP()**. To delete all templates for the specified user ID, use **RemoveFPEx()**.

[C#]

```
Int32    iError;
SS_IDInfo idInfo;

…
idInfo = new SS_IDInfo();
idInfo.ID = Convert.ToInt32(idStr);
idInfo.FingerNumber = Convert.ToByte(FNStr);
idInfo.SampleNumber = Convert.ToByte(SNStr);

iError = m_SecuSearch.RemoveFP(idInfo);
```

[VB.NET]

```
Dim iError As Int32
Dim temp As String
Dim idInfo As SS_IDInfo

idInfo = New SS_IDInfo
idInfo.ID = Convert.ToInt32(idStr)
idInfo.FingerNumber = Convert.ToByte(FNStr)
idInfo.SampleNumber = Convert.ToByte(SNStr)
```

```
iError = m_SecuSearch.RemoveFP(idInfo)
```

5.3.10. Manage SecuSearch Engine Database

- **Clear SecuSearch Engine database**
  Deletes all fingerprint data in the SecuSearch Engine database

  [C#]
  ```
  Int32 iError;
  iError = m_SecuSearch.ClearFPDB();
  ```

  [VB.NET]
  ```
  Dim iError As Int32
  iError = m_SecuSearch.ClearFPDB()
  ```

- **Read Data Count in SecuSearch Engine database**
  Reads the count of fingerprint data in the SecuSearch Engine database

  [C#]
  ```
  Int32 iError;
  Int32 iCount = 0;

  iError = m_SecuSearch.GetFPCount(ref iCount);
  ```

  [VB.NET]
  ```
  Dim iError As Int32
  Dim iCount As Int32

  iError = m_SecuSearch.GetFPCount(iCount)
  ```

- **Read Data List in SecuSearch Engine database**
  Reads the list of fingerprint data in the SecuSearch Engine database

  [C#]
  ```
  Int32 iError;
  Int32 iCount = 20000;
  SS_CandList[] idArray;
  idArray = new SS_CandList[iCount];

  iError = m_SecuSearch.GetFPList(iCount, idArray, ref iCount);
  ```

  [VB.NET]
  ```
  Dim iError As Int32
  Dim iCount As Int32
  Dim idArray() As SS_CandList

  iCount = 20000
  ReDim idArray(iCount)

  iError = m_SecuSearch.GetFPList(iCount, idArray, iCount)
  ```

- **Read Fingerprint Data from SecuSearch Engine database**

Reads fingerprint data in the SecuSearch Engine database

[C#]

```
Int32 iError;
Byte[] minData;
minData = new Byte[400];
SS_IDInfo idInfo;

idInfo.ID = 1234;
idInfo.FingerNumber = 1;
idInfo.SampleNumber = 1;

iError = m_SecuSearch.GetFPData(idInfo, minData);
```

[VB.NET]

```
Dim iError As Int32
Dim minData(400) As Byte
Dim idInfo As SS_IDInfo

idInfo.ID = 1234
idInfo.FingerNumber = 1
idInfo.SampleNumber = 1

iError = m_SecuSearch.GetFPData(idInfo, minData)
```

Appendix A. API Reference

Structures, error codes, and declared values are defined in ‘SecuSearchDef.h.’ The API functions are defined in ‘SecuSearchAPI.h,’ which includes ‘SecuSearchDef.h.’

A.1. Functions

A.1.1. SecuSearch Engine Initialize/Terminate /Parameter set

SecuSearchAPI_InitializeEngine: SecuSearch Engine initialization

DWORD SecuSearchAPI_InitializeEngine(
IN ENGINE_PARAM_* param,
);

Description

Inputs user setting parameters of SecuSearch Engine to initialize SecuSearch Engine. This function checks the validity of the license file and allocates memory for the internal memory resident fingerprint DB and initializes global variables. Refer to ENGINE_PARAM_ structure for more information.

Parameters

param: User setting parameter

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Failed to initialize
FPS_ERROR_MEM_OVERFLOW: Failed to allocate memory.

SecuSearchAPI_TerminateEngine: Terminate SecuSearch Engine

DWORD SecuSearchAPI_TerminateEngine (void);

Description

Terminates SecuSearch Engine. To turn off 1:N matching system, this function should be called. This function frees all the memory that the SecuSearch Engine has allocated.

Parameters

Return Values

FPS_ERROR_NONE: Error None
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_GetEngineParam: Read SecuSearch Engine parameter

DWORD SecuSearchAPI_GetEngineParam(
OUT ENGINE_PARAM_* param
);

Description

Reads parameter values of the SecuSearch Engine

Parameters

param: SecuSearch Engine parameter

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_SetEngineParam: Set SecuSearch Engine parameter

```
DWORD SecuSearchAPI_SetEngineParam(
IN   ENGINE_PARAM_* param
);
```

Description
Sets SecuSearch Engine parameter. Refer to ENGINE_PARAM_ structure for more information.

Parameters
param: User setting parameter

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

A.1.2. Fingerprint registration, removal, search

SecuSearchAPI_RegisterFP: Fingerprint registration

```
DWORD SecuSearchAPI_RegisterFP(
IN   BYTE *feat_buff,
IN   ID_INFO_* user_id
);
```

Description
Inputs 400 byte fingerprint minutiae template extracted by FDx SDK or FDA0x DK and user information to register a fingerprint to the internal memory resident fingerprint DB of the SecuSearch Engine. User information consists of user ID, finger number and sample number of fingerprint. These values are referred from the User DB.

Parameters
feat_buff: 400 byte fingerprint minutiae template (from FDx SDK or FDA0x DK)
user_id: User information (user ID, finger number, sample number)

Return Values
FPS_ERROR_NONE: No error
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_OVER_LIMIT: Over the maximum number of fingerprints to register
FPS_ERROR_INVALD_TEMPLATE: Invalid template
FPS_ERROR_MEM_OVERFLOW: Failed memory allocation

SecuSearchAPI_RegisterFPEx: Fingerprint registration (FIR template)

```
DWORD SecuSearchAPI_RegisterFPEx(
IN   const char* szFIR,
IN   ID_INFO_* user_id
);
```

Description
Inputs FIR format fingerprint minutiae template (character string type) extracted by SecuBSP and user

information to register a fingerprint to the internal memory resident fingerprint DB of the SecuSearch Engine. User information consists of user ID, finger number, and sample number of fingerprint. These values are referred from the User DB.

Parameters
szFIR: Character string type FIR fingerprint minutiae template
user_id: User information (user ID, finger number, sample number)

Return Values
FPS_ERROR_NONE: No error
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_OVER_LIMIT: Over the maximum number of fingerprints to register
FPS_ERROR_INVALD_TEMPLATE: Invalid template
FPS_ERROR_MEM_OVERFLOW: Failed memory allocation

SecuSearchAPI_RegisterFPExW : Fingerprint registration (FIR template)

```
DWORD SecuSearchAPI_RegisterFPExW(
IN   BSTR bstrFIR,
IN   ID_INFO_* user_id
);
```

Description
Inputs FIR format fingerprint minutiae template (wide character type) extracted by SecuBSP and user information to register fingerprint to the internal memory resident fingerprint DB of the SecuSearch Engine. User information consists of user ID, finger number, and sample number of fingerprint. These values are referred from the User DB.

Parameters
bstrFIR: Wide character type FIR fingerprint minutiae template
user_id: User information (user ID, finger number, sample number)

Return Values
FPS_ERROR_NONE: No error
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_OVER_LIMIT: Over the maximum number of fingerprints to register
FPS_ERROR_INVALD_TEMPLATE: Invalid template
FPS_ERROR_MEM_OVERFLOW: Failed memory allocation

SecuSrchAPI_GetFIRInfo : Read FIR information

```
DWORD SecuSrchAPI_GetFIRInfo(
IN   const char* szFIR,
OUT          FIR_INFO_* pFIRInfo
);
```

Description
The FIR format fingerprint minutiae template extracted by SecuBSP contains information about one or more fingerprints per user. Call this function to get the information about finger numbers and sample numbers from the FIR template. Using this information, the application programmer can add fingerprint information in the User DB.

Parameters
szFIR: Character string type FIR template (from SecuBSP)
pFIRInfo: All the fingerprint information (finger number, sample number) in the FIR

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSrchAPI_GetFIRInfoW: Read FIR information

DWORD SecuSrchAPI_GetFIRInfoW(
IN BSTR bstrFIR,
OUT FIR_INFO_* pFIRInfo
);

Description

The FIR format fingerprint minutiae template extracted by SecuBSP contains information about one or more fingerprints per user. Call this function to get the information about finger numbers and sample numbers from the FIR template. Using this information, the application programmer can add fingerprint information in the User DB.

Parameters

bstrFIR: Wide character type FIR template
pFIRInfo: All the fingerprint information (finger number, sample number) in the FIR

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_RemoveFP: Template removal

DWORD SecuSearchAPI_RemoveFP(
IN ID_INFO_* user_id
);

Description

Removes one template from the internal DB of the SecuSearch Engine. First, get fingerprint information (user ID, finger number, sample number) by referring to the User DB, and call this function to remove the corresponding fingerprint data from the SecuSearch DB.

Parameters

user_id: User information (user ID, finger number, sample number)

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_RemoveFPEx: Template removal

DWORD SecuSearchAPI_RemoveFPEx(
IN ID_INFO_* user_id
);

Description

Removes all templates for the specified user ID from the internal DB of the SecuSearch Engine. First, get fingerprint information (user ID, finger number, sample number) by referring to the User DB, and call this function to remove the corresponding fingerprint data from the SecuSearch DB.

Parameters

user_id: User information (only user ID is needed; finger number/ sample number will be ignored)

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_SearchFP: Similar fingerprint candidates search

```
DWORD SecuSearchAPI_SearchFP(
IN    BYTE *feat_buff,
OUT           CAND_LIST_ *candidate,
OUT           DWORD*  count_candidate
);
```

Description
Searches for fingerprint candidates that are similar to the input (query) fingerprint. If a 400 byte fingerprint minutiae template extracted by FDx SDK or FDA0x DK and an empty ID_INFO structure (result) are input, this function returns a list of similar fingerprint candidates. The candidates are listed in descending order of confidence level that quantifies the degree of similarity between the compared fingerprints. If the confidence level is above 5, it can be regarded that the two fingerprints match. The list contains the registered fingerprint information (user ID, finger number, sample number). The number of candidates in the list can be set by calling ***SecuSearchAPI_SetParameter()*** .

Parameters
feat_buff: A 400 byte fingerprint minutiae template (from FDx SDK or FDA0x DK)
candidate: List of similar fingerprint candidates
count_candidate: Number of candidates returned

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_SearchFPEx: Similar fingerprint candidates search

```
DWORD SecuSearchAPI_SearchFPEx(
IN    const char* szFIR,
OUT           CAND_LIST_ *candidate,
OUT           DWORD* count_candidate
);
```

Description
Searches for fingerprint candidates that are similar to the input (query) fingerprint. If a FIR fingerprint minutiae template extracted by SecuBSP and an empty ID_INFO structure (result) are input, this function returns a list of similar fingerprint candidates. The candidates are listed in descending order of confidence level that quantifies the degree of similarity between the compared fingerprints. If the confidence level is above 5, it can be regarded that two fingerprints match. The list contains the registered fingerprint information (user ID, finger number, sample number). The number of candidates in the list can be set by calling ***SecuSearchAPI_SetParameter()***.

Parameters
szFIR: Character string type FIR template
candidate: List of similar fingerprint candidates
count_candidate: Number of candidates returned

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_SearchFPExW: Similar fingerprint candidates search

```
DWORD SecuSearchAPI_SearchFPExW(
IN   BSTR bstrFIR,
OUT          CAND_LIST_ *candidate,
OUT          DWORD* count_candidate
);
```

Description

Searches for fingerprint candidates that are similar to the input (query) fingerprint. If a FIR fingerprint minutiae template extracted by SecuBSP and an empty ID_INFO structure (result) are input, this function returns a list of similar fingerprint candidates. The candidates are listed in descending order of confidence level that quantifies the degree of similarity between the compared fingerprints. If the confidence level is above 5, it can be regarded that two fingerprints match. The list contains the registered fingerprint information (user ID, finger number, sample number). The number of candidates in the list can be set by calling ***SecuSearchAPI_SetParameter()***.

Parameters

bstrFIR: Wide character type FIR template
candidate: List of similar fingerprint candidates
count_candidate: Number of candidates returned

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_IdentifyFP: Fingerprint Identification

```
DWORD SecuSearchAPI_IdentifyFP(
IN   BYTE *feat_buff,
IN   BYTE seculevel,
OUT          ID_INFO_ *result
);
```

Description

Identifies an input (query) fingerprint. This function determines whether there is a fingerprint in the DB that matches the input fingerprint. If a 400 byte fingerprint minutiae template extracted by FDx SDK or FDA0x DK, a security level, and an empty ID_INFO structure (result) are input, this function returns FPS_ERROR_NONE and the identified fingerprint information if the input fingerprint is identified. Otherwise, this function returns FPS_ERROR_ IDENTIFICATION_FAIL. The security level is a threshold that is used to decide whether two fingerprints match. The security level value can be from 1 to 9. If the security level is high, identification will succeed only when the similarity of two fingerprints is high enough.

Parameters

feat_buff : A 400 byte fingerprint minutiae template (from FDx SDK or FDA0x DK)
seculevel: Security level (1~9); the threshold value to decide identification
result: Information of identified fingerprint in DB

Return Values

FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_NONE: Input fingerprint is identified
FPS_ERROR_IDENTIFICATION_FAIL: Input fingerprint is not identified
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_IdentifyFPEx: Fingerprint Identification

```
DWORD SecuSearchAPI_IdentifyFPEx(
IN    const char* szFIR,
IN    BYTE seculevel,
OUT           ID_INFO_ *result
);
```

Description

Identifies an input (query) fingerprint. This function determines whether there is a fingerprint in the DB that matches the input fingerprint. If a FIR format fingerprint minutiae template extracted by SecuBSP, a security level, and an empty ID_INFO structure (result) are input, this function returns FPS_ERROR_NONE and the identified fingerprint information if the input fingerprint is identified. Otherwise, this function returns FPS_ERROR_ IDENTIFICATION_FAIL. The security level is a threshold that is used to decide whether two fingerprints match. The security level value can be from 1 to 9. If the security level is high, identification will succeed only when the similarity of two fingerprints is high enough.

Parameters

feat_buff: Character string type FIR template (from SecuBSP)
seculevel: Security level (1~9); the threshold value to decide identification
result: Information of identified fingerprint in DB

Return Values

FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_NONE: Input fingerprint is identified
FPS_ERROR_IDENTIFICATION_FAIL: Input fingerprint is not identified
FPS_ERROR_INVALD_TEMPLATE: Invalid template

SecuSearchAPI_IdentifyFPExW : Fingerprint identification

```
DWORD SecuSearchAPI_IdentifyFPExW(
IN    BSTR bstrFIR,
IN    BYTE seculevel,
OUT           ID_INFO_ *result
);
```

Description

Identifies an input (query) fingerprint. This function determines whether there is a fingerprint in the DB that matches the input fingerprint. If a FIR format fingerprint minutiae template extracted by SecuBSP, a security level, and an empty ID_INFO structure (result) are input, this function returns FPS_ERROR_NONE and the identified fingerprint information if the input fingerprint is identified. Otherwise, this function returns FPS_ERROR_ IDENTIFICATION_FAIL. The security level is a threshold that is used to decide whether two fingerprints match. The security level value can be from 1 to 9. If the security level is high, identification will succeed only when the similarity of two fingerprints is high enough.

Parameters

feat_buff: Wide character type FIR template (from SecuBSP)
seculevel: Security level (1~9); the threshold value to decide identification
result: Information of identified fingerprint in DB

Return Values

FPS_ERROR_NOT_INIT: Not initialized
FPS_ERROR_NONE: Input fingerprint is identified
FPS_ERROR_IDENTIFICATION_FAIL: Input fingerprint is not identified
FPS_ERROR_INVALD_TEMPLATE: Invalid template

A.1.3. DB Management Load/ Clear

SecuSearchAPI_ClearFPDB : Clear fingerprint DB

DWORD SecuSearchAPI_ClearFPDB(void);

Description

Removes all fingerprint data in the internal memory resident fingerprint DB of SecuSearch Engine.

Parameters

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_GetFPCount : Get fingerprint Data Count

```
DWORD SecuSearchAPI_GetFPCount(
OUT     DWORD* count
);
```

Description

Reads the count of fingerprint data in the internal memory resident fingerprint DB of SecuSearch Engine.

Parameters

count: Data count stored in the internal memory

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_GetFPList : Get fingerprint Data List

```
DWORD SecuSearchAPI_GetFPList(
IN      DWORD buff_size,
OUT     ID_INFO_* idList,
OUT     DWORD* count);
```

Description

Reads the list of fingerprint data in the internal memory resident fingerprint DB of SecuSearch Engine.

Parameters

buff_size: Size of ID_INFO_ buffer to receive the data list
idList: List of fingerprint data information
count: Data count actually received

Return Values

FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

SecuSearchAPI_GetFPData : Get fingerprint Data

```
DWORD SecuSearchAPI_GetFPData(
IN      ID_INFO_* idInfo,
OUT     BYTE* feat_buff
```

);

Description
Reads fingerprint data in the internal memory resident fingerprint DB of SecuSearch Engine.

Parameters
idInfo: Data information to receive
feat_buff: A 400 byte fingerprint minutiae template stored in the internal memory

Return Values
FPS_ERROR_NONE: Error none
FPS_ERROR_NOT_INIT: Not initialized

A.2. SecuSearch Engine Data Structures

ENGINE_PARAM_
typedef struct ENGINE_PARAM

```
{
CHAR szLicenseFile[MAX_PATH];
DWORD CandidateNumber;    // 4 Bytes
DWORD MemPoolSizeMB;      // 4 Bytes
DWORD reserved1;          // 4 Bytes
DWORD reserved2;          // 4 Bytes
DWORD reserved3;          // 4 Bytes
DWORD reserved4;          // 4 Bytes
DWORD reserved5;          // 4 Bytes
DWORD * reserved6;        // 4 Bytes
} ENGINE_PARAM_;
```

Description
SecuSearch Engine parameters set by user
Members
szLicenseFile: Full path of SecuSearch license file
CandidateNumber: Number of fingerprint candidates resulting from a search
MemPoolSizeMB: Size of memory (in MB) to allocate during engine initialization
Reserved1 ~ reserved6: reserved

ID_INFO_
typedef struct ID_INFO

```
{
DWORD     ID;             // 4 Bytes
BYTE      FingerNumber;   // 1 Byte
BYTE      SampleNumber;   // 1 Byte
} ID_INFO_;
```

Description
User fingerprint information
Members
ID: User ID number in User DB
FingerNumber: Position of the finger (0~10, see Appendix A.3.2)
SampleNumber: Number of times the fingerprint has been input

CAND_LIST_

```
typedef struct CAND_LIST
{
DWORD     ID;                 // 4 Bytes
BYTE      FingerNumber;       // 1 Byte
BYTE      SampleNumber;       // 1 Byte
BYTE      ConfidenceLevel;    // 1 Byte
} CAND_LIST_;
```

Description
Fingerprint candidate list as a search result
Members
ID: User ID number (referred in User DB)
FingerNumber: Position of the finger (0~10, see Appendix A.3.2)
SampleNumber: Number of times the fingerprint has been input
ConfidenceLevel: Confidence level (similarity between input and result fingerprint) (1~9)

FIR_INFO

```
typedef struct FIR_INFO
{
BYTE      FingerCount;        // 1 Byte
FP_INFO_  FPInfo[20];
} FIR_INFO_;
```

Description
Fingerprint information in a FIR template
Total number of fingerprints in FIR, finger number, and sample number
Members
FingerCount: Total number of fingerprints in FIR
FPInfo: Information about each fingerprint (FingerNumber & SampleNumber)

FP_INFO

```
typedef struct FP_INFO
{
BYTE      FingerNumber;       // 1 Byte
BYTE      SampleNumber;       // 1 Byte
} FP_INFO_;
```

Description
Fingerprint information
Information about finger number & sample number
Members
FingerNumber: Position of the finger (0~10, see Appendix A.3.2)
SampleNumber: Number of times the fingerprint has been input

A.3. Error Codes & Defined Constants

A.3.1. Error Code

Error code	Value	Definition
FPS_ERROR_NONE	0	No error
FPS_ERROR_NOT_INIT	1	Engine is not initialized
FPS_ERROR_INVALID_PARAM	2	Invalid parameter
FPS_ERROR_MEM_OVERFLOW	101	Failed to allocate memory
FPS_ERROR_SAVE_DB	102	Failed to save DB

FPS_ERROR_LOAD_DB	103	Failed to load DB
FPS_ERROR_INVALD_TEMPLATE	104	Invalid Template
FPS_ERROR_DATA_EXIST	105	Data already exists
FPS_ERROR_UNKNOWN_DB	106	Cannot identify DB
FPS_ERROR_NO_DATA	107	No data returned
FPS_ERROR_OVER_LIMIT	201	Over the maximum number of fingerprints allowed to register (per license)
FPS_ERROR_IDENTIFICATION_FAIL	202	Input fingerprint is not identified
FPS_ERROR_LICENSE_LOAD	501	Failed to load license file
FPS_ERROR_LICENSE_KEY	502	Invalid license key
FPS_ERROR_LICENSE_EXPIRED	503	License is expired
FPS_ERROR_LICENSE_WRITE	504	Failed to create license file
FPS_ERROR_SECUSEARCH_DLL_UNLOADED	601	Failed to load SecuSearch.DLL
FPS_ERROR_SECUSEARCHAPI_DLL_UNLOADED	602	Failed to load SecuSearchAPI.DLL
FPS_ERROR_LOAD_SGFPA_DLL	701	Failed to load FDx SDK module
FPS_ERROR_LOAD_SECUBSPMX_DLL	702	Failed to load SecuBSP SDK module
FPS_ERROR_PLOCK_MEM_INVALID_PARAM	2001	Memory error
FPS_ERROR_PLOCK_MEM_NOT_INIT	2002	Memory error
FPS_ERROR_HEAP_CREATE	2003	Memory error
FPS_ERROR_HEAP_ALLOC	2004	Memory error
FPS_ERROR_HEAP_FREE	2005	Memory error
FPS_ERROR_SET_LOCK_PAGE_PRIVILEGE	2006	No privilege for Lock Pages in Memory Refer to the Distribution Guide for more information
FPS_ERROR_ALLOC_PHYS_MEM	2007	Memory error
FPS_ERROR_FREE_PHYS_MEM	2008	Memory error
FPS_ERROR_MAP_PHYS_MEM	2009	Memory error
FPS_ERROR_RESERVE_MEM	2010	Memory error
FPS_ERROR_LOW_MEM	2011	Memory error
FPS_ERROR_TOO_MANY_FEAT	2101	Memory error
FPS_ERROR_OPEN_FILE	2102	Open file error
FPS_ERROR_DATA_NOT_FOUND	2103	No data found
FPS_ERROR_INVALID_TYPE	3003	Invalid SecuBSP FIR type
FPS_ERROR_INVALID_FORMAT	3004	Invalid SecuBSP format
FPS_ERROR_INVALID_DATA	3005	Invalid SecuBSP data
FPS_ERROR_DATA_CONVERSION	3006	SecuBSP FIR data conversion failure
FPS_ERROR_UNKNOWN_VERSION	3007	Unknown SecuBSP version
FPS_ERROR_ENCRYPTED_DATA_ERROR	3008	SecuBSP FIR data encryption error
FPS_ERROR_MUST_BE_PROCESSED_DATA	3009	SecuBSP FIR data error

A.3.2. Defined constants

CONF_LEVEL_LOWEST	1
CONF_LEVEL_LOWER	2
CONF_LEVEL_LOW	3
CONF_LEVEL_BELOW_NORMAL	4
CONF_LEVEL_NORMAL	5
CONF_LEVEL_ABOVE_NORMAL	6
CONF_LEVEL_HIGH	7
CONF_LEVEL_HIGHER	8
CONF_LEVEL_HIGHEST	9
FIN_NUM_UNKNOWN	0
FIN_NUM_RIGHT_THUMB	1

FIN_NUM_RIGHT_INDEX	2
FIN_NUM_ RIGHT_MIDDLE	3
FIN_NUM_ RIGHT_RING	4
FIN_NUM_ RIGHT_LITTLE	5
FIN_NUM_ LEFT_LITTLE	6
FIN_NUM_ LEFT_INDEX	7
FIN_NUM_ LEFT_MIDDLE	8
FIN_NUM_ LEFT_RING	9
FIN_NUM_ LEFT_LITTLE	10

Appendix B. .NET Library Reference

B.1. SecuSearch .NET Class Reference

B.1.1. InitializeEngine

[C++]
```
Int32 InitializeEngine(SS_EngineParam* param);
```

[VB.NET]
```
Public Function InitializeEngine(ByVal param As SecuGen.SecuSearchSDK.SS_EngineParam) As Integer
```

B.1.2. TerminateEngine

[C++]
```
Int32 TerminateEngine(void);
```

[VB.NET]
```
Public Function TerminateEngine() As Integer
```

B.1.3. GetLastError

[C++]
```
Int32 GetLastError()
```

[VB.NET]
```
Public Function GetLastError() As Integer
```

B.1.4. SetEngineParam

[C++]
```
Int32 SetEngineParam(SS_EngineParam* param);
```

[VB.NET]
```
Public Function SetEngineParam(ByVal param As SecuGen.SecuSearchSDK.SS_EngineParam) As Integer
```

B.1.5. GetEngineParam

[C++]
```
Int32 GetEngineParam(SS_EngineParam* param);
```

[VB.NET]
```
Public Function GetEngineParam(ByVal param As SecuGen.SecuSearchSDK.SS_EngineParam) As Integer
```

B.1.6. RegisterFP

[C++]
```
Int32 RegisterFP(Byte minData[], SS_IDInfo* userID);
```

[VB.NET]
```
Public Function RegisterFP(ByVal minData() As Byte, ByVal userID As
SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.7. RegisterFPEx

[C++]
```
Int32 RegisterFPEx(String* firData, SS_IDInfo* userID);
```

[VB.NET]
```
Public Function RegisterFPEx(ByVal firData As String, ByVal userID As
SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.8. SearchFP

[C++]
```
Int32 SearchFP(Byte minData[], SS_CandListArray  candArray, Int32 reserved0);
```

[VB.NET]
```
Public Function SearchFP(ByVal minData() As Byte, ByVal candArray() As
SecuGen.SecuSearchSDK.SS_CandList, ByVal reserved0 As Integer) As Integer
```

B.1.9. SearchFPEx

[C++]
```
Int32 SearchFPEx(String* firData, SS_CandListArray  candArray, Int32 reserved0);
```

[VB.NET]
```
Public Function SearchFPEx(ByVal firData As String, ByVal candArray() As
SecuGen.SecuSearchSDK.SS_CandList, ByVal reserved0 As Integer) As Integer
```

B.1.10. IdentifyFP

[C++]
```
Int32 IdentifyFP(Byte minData[], unsigned char secuLevel, SS_IDInfo* idInfo);
```

[VB.NET]
```
Public Function IdentifyFP(ByVal minData() As Byte, ByVal secuLevel As Byte, ByVal idInfo
As SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.11. IdentifyFPEx

[C++]
```
Int32 IdentifyFPEx(String* firData, unsigned char seculevel, SS_IDInfo* idInfo);
```

[VB.NET]
```
Public Function IdentifyFPEx(ByVal firData As String, ByVal seculevel As Byte, ByVal
```

```
idInfo As SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.12. RemoveFP

[C++]
```
Int32 RemoveFP(SS_IDInfo* userID);
```

[VB.NET]
```
Public Function RemoveFP(ByVal userID As SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.13. RemoveFPEx

[C++]
```
Int32 RemoveFPEx(SS_IDInfo* userID);
```

[VB.NET]
```
Public Function RemoveFPEx(ByVal userID As SecuGen.SecuSearchSDK.SS_IDInfo) As Integer
```

B.1.14. GetFIRInfo

[C++]
```
Int32 GetFIRInfo(String* firData, SS_FIRInfo* pFIRInfo)
```

[VB.NET]
```
Public Function GetFIRInfo(ByVal firData As String, ByVal pFIRInfo As
SecuGen.SecuSearchSDK.SS_FIRInfo) As Integer
```

B.1.15. ClearFPDB

[C++]
```
Int32 ClearFPDB();
```

[VB.NET]
```
Public Function ClearFPDB() As Integer
```

B.1.16. GetFPCount

[C++]
```
Int32 GetFPCount(ref int);
```

[VB.NET]
```
Public Function GetFPCount(ByRef Integer) As Integer
```

B.1.17. GetFPList

[C++]
```
Int32 GetFPList(int, SecuGen.SecuSearchSDK.SS_CandList[], ref int);
```

[VB.NET]
```
Public Function GetFPList(ByVal Integer, ByVal SecuGen.SecuSearchSDK.SS_CandList(), ByRef
```

```
Integer) As Integer
```

B.1.18. GetFPData

```
[C++]
Int32 GetFPData(SecuGen.SecuSearchSDK.SS_IDInfo, byte[]);

[VB.NET]
Public Function GetFPData(ByVal SecuGen.SecuSearchSDK.SS_IDInfo, ByVal Byte()) As Integer
```

B.2. SS_EngineParam

```
[C++]
public __gc class SS_EngineParam
{
   public:
            Int32 CandidateNumber;
            Int32 reserved0;
            Int32 reserved1;
            Int32 reserved2;
            Int32 reserved3;
            Int32 reserved4;
            Int32 reserved5;
            IntPtr reserved6;
        SS_EngineParam()
        {
            CandidateNumber = 20000;
        }
};
```

B.3. SS_CandList

```
[C++]
value public struct SS_CandList
{
  //public:
        Int32 ID;
        BYTE    FingerNumber;
        BYTE    SampleNumber;
    BYTE        ConfidenceLevel;
    SS_CandList()
    {
        ID = 0;
            FingerNumber = 0;
            SampleNumber = 0;
        ConfidenceLevel = 0;
    }
};
```

B.4. SS_IDInfo

```
[C++]
public __gc class SS_IDInfo
{
   public:
            Int32 ID;
            Byte FingerNumber;
            Byte SampleNumber;
        SS_IDInfo()
        {
```

```
            ID = 0;
                FingerNumber = 0;
                SampleNumber = 0;
        }
};
```

B.5. SS_FIRInfo

```
[C++]
public __gc class SS_FIRInfo
{
  public:
        Byte        FingerCount;
    SS_InfoArray FPInfoArray;
             SS_FIRInfo()
    {
        FPInfoArray = new SS_FPInfo[20];
    }
};

__value public struct SS_FPInfo
{
        Byte FingerNumber;
        Byte SampleNumber;
};
typedef SS_FPInfo SS_InfoArray[];
```

B.7. SS_Error Enumeration

```
[C++]
public __value enum SS_Error
{
        ERROR_NONE                          = 0,
        ERROR_NOT_INIT                      = 1,
        ERROR_INVALID_PARAM                 = 2,
        ERROR_MEM_OVERFLOW                  = 101,
        ERROR_SAVE_DB                       = 102,
        ERROR_LOAD_DB                       = 103,
        ERROR_INVALD_TEMPLATE               = 104,
        ERROR_DATA_EXIST                    = 105,
        ERROR_UNKNOWN_DB                    = 106,
        ERROR_NO_DATA_FOUND                 = 107,
        ERROR_OVER_LIMIT                    = 201,
        ERROR_IDENTIFICATION_FAIL           = 202,
        ERROR_LICENSE_LOAD                  = 501,
        ERROR_LICENSE_KEY                   = 502,
        ERROR_LICENSE_EXPIRED               = 503,
        ERROR_LICENSE_WRITE                 = 504,
        ERROR_SECUSEARCH_DLL_UNLOADED       = 601,
        ERROR_SECUSEARCHAPI_DLL_UNLOADED    = 602,
        ERROR_LOAD_SGFPA_DLL                = 701,
        ERROR_LOAD_SECUBSPMX_DLL            = 702,

        // Memory
        ERROR_PLOCK_MEM_INVALID_PARAM       = 2001,
        ERROR_PLOCK_MEM_NOT_INIT            = 2002,
        ERROR_HEAP_CREATE                   = 2003,
        ERROR_HEAP_ALLOC                    = 2004,
        ERROR_HEAP_FREE                     = 2005,
        ERROR_SET_LOCK_PAGE_PRIVILEGE       = 2006,
        ERROR_ALLOC_PHYS_MEM                = 2007,
        ERROR_FREE_PHYS_MEM                 = 2008,
        ERROR_MAP_PHYS_MEM                  = 2009,
        ERROR_RESERVE_MEM                   = 2010,
        ERROR_LOW_MEM                       = 2011,

        ERROR_TOO_MANY_FEAT                 = 2101,
        ERROR_OPEN_FILE                     = 2102,
```

```
        ERROR_DATA_NOT_FOUND                = 2103,

        //SecuBSP Error
        ERROR_INVALID_TYPE                  = 3003,
        ERROR_INVALID_FORMAT                = 3004,
        ERROR_INVALID_DATA_FOUND            = 3005,
        ERROR_DATA_CONVERSION               = 3006,
        ERROR_UNKNOWN_VERSION               = 3007,
        ERROR_ENCRYPTED_DATA_ERROR          = 3008,
        ERROR_MUST_BE_PROCESSED_DATA        = 3009,
};
```

B.8. SS_ConfLevel Enumeration

```
[C++]
public __value enum SS_ConfLevel // Confidence Level
{
    LOWEST              = 1,
    LOWER               = 2,
    LOW                 = 3,
    BELOW_NORMAL        = 4,
    NORMAL              = 5,
    ABOVE_NORMAL        = 6,
    HIGH                = 7,
    HIGHER              = 8,
    HIGHEST             = 9,
};
```

B.9. SS_FingerNumber Enumeration

```
[C++]
public __value enum SS_FingerNumber // FingerNumber
{
    UNKNOWN             = 0,
    RIGHT_THUMB         = 1,
    RIGHT_INDEX         = 2,
    RIGHT_MIDDLE        = 3,
    RIGHT_RING          = 4,
    RIGHT_LITTLE        = 5,
    LEFT_THUMB          = 6,
    LEFT_INDEX          = 7,
    LEFT_MIDDLE         = 8,
    LEFT_RING           = 9,
    LEFT_LITTLE         = 10,
};
```