



SecuBSP® SDK *Pro* Manual

Windows

Secure Biometric Service Provider for applications using SecuGen® fingerprint readers

SG1-0020B-004 (06/09)

© Copyright 1998-2009 SecuGen Corporation.

ALL RIGHTS RESERVED. Specifications are subject to change without notice. SecuGen, Auto-On, FDP02, FDU01, FDU02, FDU03, FDU04, SDU03, SecuAPI and SecuBSP are registered trademarks of SecuGen Corporation. All other brands or products may be trademarks, service marks or registered trademarks of their respective owners.

Contents

BEFORE YOU BEGIN	VI
BIOMETRICS OVERVIEW	VI
ABOUT SECUGEN	VI
ABOUT SECUGEN PRODUCTS	VII
CHAPTER 1. INTRODUCTION	8
1.1. FEATURES	8
1.2. DEVELOPMENT MODEL	8
1.3. BIOMETRIC FUNCTIONS	10
1.3.1. Low-Level API Functions	10
1.3.2. High-Level API Functions	10
1.4. FIR - FINGERPRINT IDENTIFICATION RECORD	11
1.5. TERMINOLOGY	11
CHAPTER 2. INSTALLATION	12
2.1. SYSTEM REQUIREMENTS	12
2.2. INSTALLATION	12
2.3. DIRECTORIES & FILES	13
2.3.1. Windows System Directory	13
2.3.2. Inc: Function Header File	13
2.3.3. Lib: Library File (/Lib/x64 for 64-bit Library File)	13
2.3.4. Bin: Runtime modules (/Bin/x64 for 64-bit Runtime modules)	13
2.3.5. Samples: Sample Programs	13
2.3.6. .NET Framework 2.0 Redistributable	14
2.4. DEMO PROGRAM	15
2.4.1. BSP Information	16
2.4.2. Device Functions	17
2.4.3. Enroll	17
2.4.4. Verify	17
CHAPTER 3. SECUBSP PROGRAMMING IN C/C++	18
3.1. CHECK MODULE VALIDITY	18
3.2. MODULE INITIALIZATION & TERMINATION	19
3.2.1. Initialize Module	19
3.2.2. Terminate Module	19
3.3. DEVICE FUNCTIONS	20
3.3.1. Enumerating Devices	20
3.3.2. Device Initialization	21
3.3.3. Closing Device	21
3.3.4. Getting Device Information	21
3.3.5. Using Auto-On	22
3.4. FINGERPRINT ENROLLMENT	23
3.4.1. Retrieving the FIR	23
3.4.2. Converting the FIR into a Binary Stream	24
3.4.3. Retrieving the Text-Encoded FIR	24
3.5. VERIFICATION	26
3.5.1. Verification with the FIR Handle	26
3.5.2. Verification with the FIR	26
3.5.3. Verification with Text-Encoded FIR	27
3.6. THE CLIENT/SERVER ENVIRONMENT	28

3.6.1. Capturing Fingerprint Data.....	29
3.6.2. Verification on a Server System.....	29
3.7. PAYLOAD.....	30
3.7.1. Inserting Payload.....	30
3.7.2. Retrieving Payload from the Template.....	31
3.8. LOAD RESOURCE FILE.....	32
3.9. DATA CONVERSION FUNCTIONS.....	32
3.9.1. Importing FDx Data.....	32
3.9.2. Exporting FDx Data.....	32
3.10. AUDIT DATA.....	33
3.11. ANSI378 FORMAT FINGERPRINT DATA.....	34
3.11.1. Retrieving ANSI378 format FIR data.....	34
3.11.2. Restrictions on Using ANSI378 Format.....	34
CHAPTER 4. SECUBSP COM PROGRAMMING.....	35
4.1. MODULE INITIALIZATION AND CLOSURE.....	35
4.1.1. Initializing the Module.....	35
4.1.2. Terminate the module use.....	35
4.2. DEVICE RELATED PROGRAMMING.....	36
4.2.1. Listing Devices.....	36
4.2.2. Initializing the Device.....	37
4.2.3. Closing the Device.....	38
4.2.4. Using Auto-On.....	38
4.3. FINGERPRINT ENROLLMENT.....	39
4.4. FINGERPRINT VERIFICATION.....	40
4.5. CLIENT/SERVER ENVIRONMENT PROGRAMMING.....	41
4.5.1. Fingerprint Enrollment.....	41
4.5.2. Fingerprint Verification.....	41
4.5.3. Fingerprint Enrollment with Stored Fingerprint.....	42
4.6. USING PAYLOAD.....	43
4.6.1. Inserting Payload into FIR.....	43
4.6.2. Extracting Payload from FIR.....	44
4.7. LOADING THE SECUBSP RESOURCE FILE.....	46
4.8. CHANGING THE WINDOW STYLE.....	46
4.9. ANSI378 FORMAT FINGERPRINT DATA.....	47
CHAPTER 5. SECUBSP COM PROGRAMMING IN ASP.....	48
5.1. REGISTRATION.....	48
5.1.1. Code for Setting Object.....	48
5.1.2. Form for Transferring Fingerprint Information.....	48
5.1.3. JavaScript Code for Fingerprint Registration.....	49
5.1.4. Storing Fingerprint Information.....	50
5.2. VERIFICATION.....	51
5.2.1. Code for Setting Object.....	51
5.2.2. Form for Transferring Fingerprint Information.....	51
5.2.3. JavaScript Code for Capturing Fingerprints.....	51
5.2.4. Matching against an Existing Fingerprint.....	53
CHAPTER 6. SECUBSP .NET PROGRAMMING.....	54
6.1. SECUBSPMX.NET.DLL AND NAMESPACE.....	54
6.1.1. Include SecuBSPMx.NET.DLL.....	54
6.1.2. Namespace.....	54
6.2. CREATE SECUBSP CLASS.....	54
6.3. DEVICE FUNCTIONS.....	55
6.3.1. Opening the device.....	55

6.3.2. Enumerate devices	56
6.3.3. Closing the Device	57
6.3.4. Using Auto-On	57
6.4. FINGERPRINT ENROLLMENT	58
6.5. FINGERPRINT VERIFICATION	58
6.6. CLIENT/SERVER ENVIRONMENT PROGRAMMING	59
6.6.1. Fingerprint Enrollment	59
6.6.2. Capturing a Fingerprint	59
6.6.3. Verifying a Fingerprint	60
6.7. USING PAYLOAD	60
6.7.1. Inserting Payload into FIR	60
6.7.2. Extracting Payload from FIR	61
6.8. LOADING THE SECUBSP RESOURCE FILE	62
6.9. CHANGING THE WINDOW STYLE	62
6.9.1. EnrollWindowOption	62
6.9.2. CaptureWindowOption	62
6.10. DATA CONVERSION FUNCTIONS	63
6.10.1. Importing FDx Data	63
6.10.2. Exporting FDx Data	63
6.11. AUDIT DATA	64
6.12. ANSI378 FORMAT FINGERPRINT DATA	65
APPENDIX A. SECUBSP COM REFERENCE	66
A.1. PROPERTY	66
A.2. METHOD	68
APPENDIX B. SECUBSP .NET REFERENCE	70
B.1. SECUBSPMX CLASS	70
B.1.1. Constructor	70
B.1.2. Properties	70
B.1.3. Methods	72
B.2. BSPINITINFO CLASS	77
B.2.1. Constructor	77
B.2.2. Fields	77
B.3. DEVICEINFO CLASS	78
B.3.1. Constructor	78
B.3.2. Fields	78
B.4. EXPORTIMAGEDATASTRUCT CLASS	78
B.4.1. Constructor	78
B.4.2. Fields	78
B.4.3. FINGER_DATA_STRUCT	79
B.5. EXPORTMINUTIAESTRUCT CLASS	79
B.5.1. Constructor	79
B.5.2. Fields	79
B.5.3. FINGER_DATA_STRUCT	80
B.6. FIRINFORMATION CLASS	80
B.6.1. Constructor	80
B.6.2. Fields	80
B.7. WINDOWOPTION CLASS	81
B.7.1. Property	81
B.8. CONSTANTS AND ENUMERATION	81
B.8.1. BSPError Enumeration	81
B.8.2. DriverEvent Enumeration	82
B.8.3. DriverMessage Enumeration	83
B.8.4. FingerIDEnumeration	83

B.8.5. <i>FDxMinType Enumeration</i>	83
B.8.6. <i>FIRDataType Enumeration</i>	83
B.8.7. <i>FIRPurpose Enumeration</i>	84
B.8.8. <i>SecurityLevel Enumeration</i>	84
B.8.9. <i>WindowStyle Enumeration</i>	84
B.8.10. <i>FIRFormat Enumeration</i>	85
APPENDIX C. USING THE WIZARDS	86
C.1. ENROLLMENT WIZARD	86
C.2. VERIFICATION WIZARD	90
C.3. BRIGHTNESS ADJUSTMENT WIZARD	90

Before You Begin

Biometrics Overview

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

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

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

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

Advantages of Using Fingerprints

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

About SecuGen

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

About SecuGen Products

SecuGen Sensor Qualities

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

Advantages of SecuGen Sensors Over Other Optical Sensors

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

Advantages SecuGen Sensors Over Semiconductor (Capacitive) Sensors

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

Strengths of SecuGen Software and Algorithms

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

Chapter 1. Introduction

SecuBSP® is the Secure Biometrics Service Provider developed by SecuGen. The SecuBSP SDK *Pro* provides feature-rich, high-level functionality that can be integrated into any application requiring fingerprint authentication. SecuBSP technology is built on SecuGen's SecuAPI® specification, working seamlessly with the most durable, compact, and reliable optical fingerprint readers in the world.

All SecuBSP SDK components contain the APIs needed for biometric authentication of multiple users and device functions. SecuBSP is equipped with self-contained User Interfaces (wizards) for enrollment and verification, enabling software application developers to quickly and easily integrate fingerprint authentication into the application of their choice.

1.1. Features

- **Optimized Graphical User Interface (Wizard)**
SecuBSP SDK *Pro* offers an excellent user interface for acquiring high-quality fingerprint images from SecuGen's advanced fingerprint readers
- **Multiple Fingerprint Enrollment**
Each user can enroll up to ten fingerprints, while one template is used to store all fingerprint data
- **Secure Fingerprint Data**
As default, SecuBSP converts fingerprint data into SecuGen's standard format (template) that is encrypted with a 128-bit encryption algorithm to protect the template from forgery or tampering by unauthorized users. If the ANSI-INCITS 378-2004 Finger Minutiae Format ("ANSI378" template) is used instead of the default, then the template is not encrypted.
- **Device Independent**
SecuBSP SDK *Pro* supports all of SecuGen's fingerprint readers seamlessly with a common programming approach for all readers
- **MINEX Compliant Algorithms**
SecuBSP SDK *Pro* uses extraction and matching algorithms that support the ANSI-INCITS 378-2004 finger minutiae format standard (for fingerprint templates). These algorithms have been tested by the National Institute of Standards and Technology (NIST) in the Ongoing MINEX program and were determined to be MINEX Compliant (<http://fingerprint.nist.gov/MINEX/>). They are listed on the GSA FIPS 201 APL as:
 - SecuGen ANSI INCITS 378 Template Generator v3.5 (Feature Extraction Algorithm)
 - SecuGen ANSI INCITS 378 Template Matcher v3.5 (Matching Algorithm)

1.2. Development Model

The core module of SecuBSP SDK *Pro* is **SecuBSPMx.DLL**. Built on SecuGen's SecuAPI, the SecuBSPMx.DLL core module implements all biometric functions. Generally, SecuBSPMx.DLL can be used with almost any 32bit compiler, but developers using Microsoft Visual Basic or Borland Delphi or similar development environments will require an ActiveX component or COM module to simplify the development process. For this reason, SecuBSP SDK *Pro* also provides the SecuBSP COM module (SecuBSPMxCOM.DLL). This component is designed for web developers and for those using RAD such as Visual Basic or Delphi.

The SecuBSP COM module uses SecuBSPMx.DLL, but it does not support all SecuBSP functions. Fingerprint data type is offered only as a text type.

Main modules of SecuBSP SDK Pro

SecuBSPMx.DLL

This is the main module of SecuBSP SDK Pro that implements all of SecuGen's biometric functions.

SgFpaMx.DLL

This is the extraction and matching module of SecuBSP SDK Pro.

SecuBSPMxCOM.DLL

SecuBSP COM module is based on Microsoft COM Technology, which facilitates easy integration of SecuBSP by developers using web development or RAD tools.

SecuBSPMx.NET.DLL

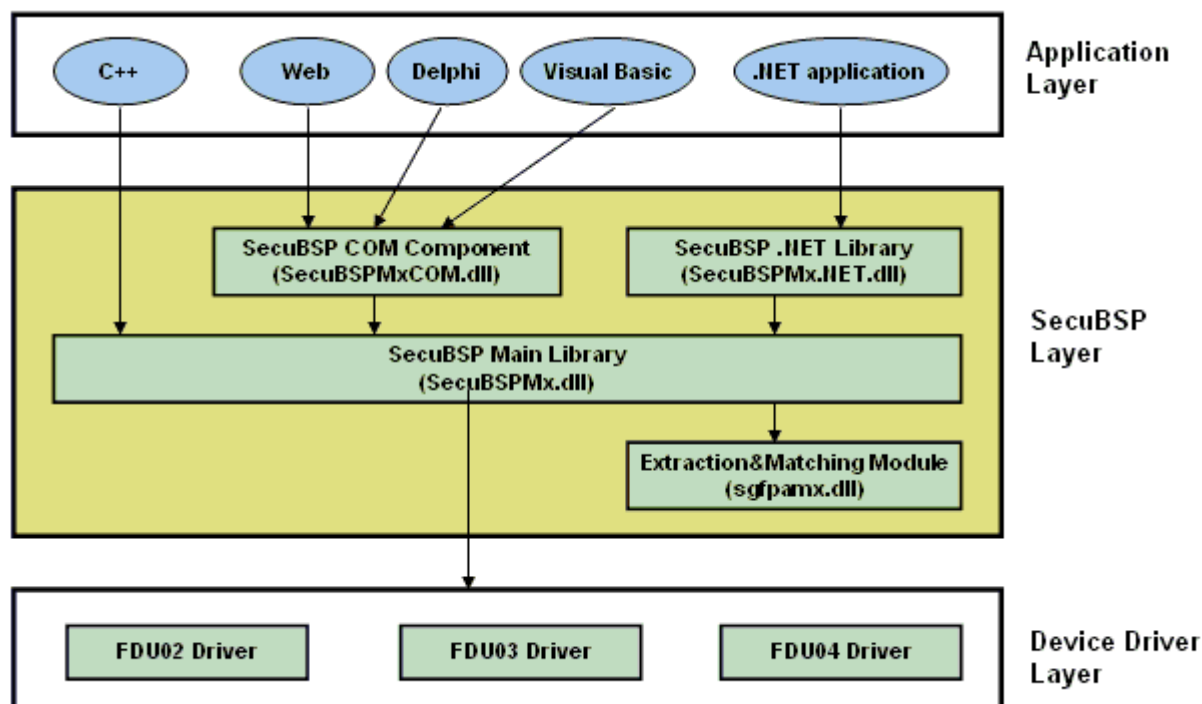
SecuBSP .NET Class library is designed for the .NET application developer to easily use SecuBSP functions in the .NET environment.

Resource DLLs

External resource DLLs can be loaded for additional language support. The SecuBSP SDK Pro is provided in English.

The following diagram shows how developers can use the modules provided in SecuBSP SDK Pro.

Development model using SecuBSP SDK Pro



1.3. Biometric Functions

SecuBSP SDK *Pro* is based on the SecuAPI specification from SecuGen and provides an advanced fingerprint authentication model for developing applications. SecuAPI is composed of two types of Biometric APIs, namely Low-level APIs and High-level APIs. Most programming requirements are satisfied by high-level APIs, which are generally used for stand-alone type applications (not for client/server or web environments). For more complex applications, such as those found in client/server computing environments, low-level APIs may sometimes be required. For example, an application captures fingerprint data on the client while users are verified and templates stored on the server. **Note:** SecuAPI high-level APIs are implemented using the SecuAPI low-level APIs.

1.3.1. Low-Level API Functions

Capture

The Capture function is used to capture a fingerprint image from a fingerprint reader, and then to extract feature points (minutiae) to form a usable fingerprint template. Multiple samples are captured for the purpose of enrollment (registration), verification, or identification. Once the capturing process is complete, the Capture function returns a Fingerprint Identification Record (FIR) as the result. The application specifies the purpose of the capture operation (enrollment, verification, or identification), and this purpose is recorded in the header of the constructed FIR.

Process

The Process function extracts feature points from the captured fingerprint image for enrollment, verification, or identification. In SecuBSP, the Capture function performs minutiae extraction; for this reason, the Process function is usually not required in the general user enrollment process.

VerifyMatch

The VerifyMatch function matches the newly input FIR against the fingerprint data in a previously stored FIR; the results of the comparison are returned.

CreateTemplate

The CreateTemplate function processes fingerprint image data to construct an enrollment template, and takes either intermediate or processed FIR as input. The CreateTemplate function can also take an old template to construct a new template, and allows a Payload to be wrapped in the new FIR.

1.3.2. High-Level API Functions

Enroll

The Enroll function is used to extract feature points from captured fingerprint images using SecuGen fingerprint readers. Enroll can also take an old FIR to construct a new FIR, and allows a Payload to be wrapped in the new FIR.

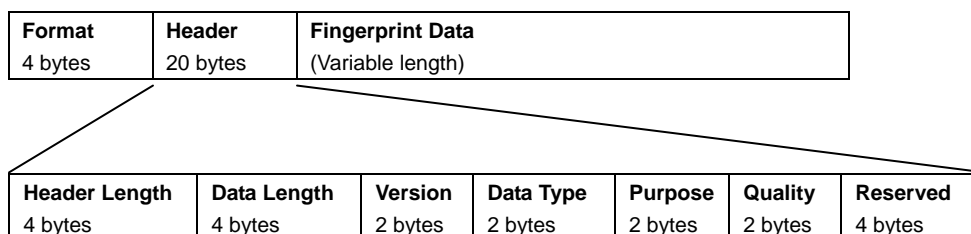
Verify

The Verify function is used to match a newly captured fingerprint sample against the previously stored fingerprint FIR; the results of the comparison are returned. If a Payload is recorded in the header of the stored template FIR, and fingerprint verification is successful, the Payload is also returned.

1.4. FIR - Fingerprint Identification Record

Fingerprint data processed in the SecuBSP is represented in a Fingerprint Identification Record (FIR) format. The FIR can include all types of fingerprint data, including raw image, intermediate data, or minutiae data. The FIR is composed of three fields: Format, Header, and Fingerprint Data.

FIR Structure



Format

The format field is 4 bytes in length and indicates the format of the fingerprint data.

Header

The header field is 20 bytes in length and is comprised of the following sub-fields:

Header Length	Indicates the size (in bytes) of the Header
Data Length	Indicates the size (in bytes) of the fingerprint data in the FIR
Version	Indicates the FIR version
Data Type	Indicates the type of fingerprint data stored in the FIR.
Purpose	Specifies the purpose of the FIR (e.g., enrollment, verification, or identification)
Quality	Indicates the quality value of the fingerprint data with a scale of 0 to 100 (future version)
Reserved	Reserved for later use

Fingerprint Data

The Fingerprint Data field contains fingerprint data. This field has a variable length, which can be affected by the values in the format field. The size of the fingerprint data is stored in the Data Length field of header.

1.5. Terminology

BSP	Biometric Service Provider; the execution module that interfaces fingerprint readers and fingerprint recognition algorithms with the developer's application
FIR	Fingerprint Identification Record (see section 1.4 for more information)
Payload	Data, such as a cryptographic key, password or user ID, stored in a FIR and released only upon verification (see section 3.7 for more information)
Sample	FIR data used for the purpose of verification
SecuBSP	Secure Biometric Service Provider; SecuGen's BSP
Template	FIR data used for the purpose enrollment

Chapter 2. Installation

SecuBSP runs on any Windows operating system, but because it is designed to operate with SecuGen® fingerprint readers, it is important to understand the hardware requirements for the USB and parallel port readers. Operating systems that do not support the USB connection protocol (e.g. Windows 95 and Windows NT) are restricted to using parallel port fingerprint readers.

2.1. System Requirements

- **OS:** Any Windows operating system*
- **CPU:** Pentium processor or later
- **Device:** Any SecuGen fingerprint reader
- **For Web development:**
 - Web server:** IIS 4.0 or above
 - Web browser:** IE 5.0 or above
- **For .NET development:**
 - .NET Framework version 2.0 or higher
 - .NET Framework SDK version 2.0 or higher

* Fingerprint readers have separate system requirements as follows:

SecuGen USB Fingerprint Reader Requirements

- USB 1.1 port (USB 2.0 port is required for FDU04-based readers)
- Microsoft Windows 98 SE/ME/2000/XP/2003/Vista
- Supported Core Fingerprint Sensors: FDU02, FDU03, FDU04, SDU03

SecuGen Parallel Port Fingerprint Reader Requirements

- 1 Parallel port (EPP mode is preferred)
- 1 PS/2 port
- Microsoft Windows 95/98/ME/NT 4.0/2000/XP/2003
- Supported Core Fingerprint Sensor: FDP02

2.2. Installation

1. Insert SecuGen SDK Collection CD into disk drive.
2. Execute setup.exe in the "SecuBSP SDK Pro for Windows" root directory of the disk.
3. Read all information displayed in the setup screens, and follow the instructions.
4. Click **NEXT** to continue
5. Click **YES** if you agree to the Software License Agreement. If you do not agree, click NO and the installation will be aborted.
6. Select a destination folder, then click **NEXT**. (The default destination is C:\Program Files\SecuGen\SecuBSP SDK Pro.)

2.3. Directories & Files

After the SecuBSP installation is complete, the following files are automatically copied to the designated directories.

2.3.1. Windows System Directory

- **SecuBSPMx.DLL** - Main module
- **Sgfpamx.DLL** – Extraction & matching module
- **SecuBSPMxCOM.DLL** - SecuBSP COM module. This DLL is automatically registered on the system registry while running **Setup**
- **SecuBSPMx.NET.DLL** - SecuBSP .NET class library.

2.3.2. Inc: Function Header File

- **SecuAPI.h** - Declarations for function prototypes and structures used in the SDK
- **SecuAPI_Basic.h** - Declarations of basic types used in SecuBSP
- **SecuAPI_Error.h** - Declarations of error constants used in SecuBSP
- **SecuAPI_Type.h** - Declarations of types used in SecuBSP
- **SecuAPI_CheckValidity.h** - Function prototype used to check module validity
- **SecuAPI_Export.h** – Declarations of FIR export and import related function prototypes in SecuBSP
- **SecuAPI_ExportType.h** – Declarations of FIR export and import related types used in SecuBSP

2.3.3. Lib: Library File (/Lib/x64 for 64-bit Library File)

- **SecuBSPMx.lib** - SecuBSP library file (links SecuBSP module statically)
- **SecuBSP_CheckValidity.lib** - Module for establishing the validity of SecuBSP module

2.3.4. Bin: Runtime modules (/Bin/x64 for 64-bit Runtime modules)

SecuBSP Runtime modules

- **SecuBSPMx.DLL** - Main module. All application using SecuBSP SDK should have this DLL.
- **Sgfpamx.DLL** – Extraction & matching module. All application using SecuBSP SDK should have this DLL.
- **SecuBSPMxCOM.DLL** - SecuBSP COM module. This DLL is automatically registered on the system registry while running **Setup**
- **SecuBSPMx.NET.DLL** - SecuBSP .NET class library.
- **objSecuBSPMx.cab** - Cabinet files to install SecuBSPMxCOM.DLL; used when downloading SecuBSPMxCOM.DLL from a web site

2.3.5. Samples: Sample Programs

Samples using SecuBSPMx.DLL

- **Samples\VisualC++** - Sample program written in Visual C++ that demonstrates the use of biometric functions in SecuBSPMx.DLL
- **Samples\AdvancedPgms\ExportImage** - Sample program written in Visual C++ that demonstrates the use of audit data in SecuBSPMx.DLL
- **Samples\AdvancedPgms\FDxToSecuBSP** - Sample program written in Visual C++ that demonstrates the fingerprint data conversion function in SecuBSPMx.DLL (from FDx 400 byte minutiae format

(SG400) to SecuBSP FIR format)

- **Samples\AdvancedPgms\SecuBSPToFDx** - Sample program written in Visual C++ that demonstrates the fingerprint data conversion function in SecuBSPMx.DLL (from SecuBSP FIR format to FDx 400 byte minutiae format (SG400))

Samples using SecuBSPMxCOM.DLL

- **Samples\VisualBasic** - Sample program written in Visual Basic that demonstrates the use of biometric functions in the SecuBSP COM control (SecuBSPMxCOM.DLL) in a Visual Basic application
- **Samples\Delphi** - Sample program written in Delphi that demonstrates the use of biometric functions in the SecuBSP COM control (SecuBSPMxCOM.DLL) in a Delphi application
- **Samples\Html** - Sample program that demonstrates the use of biometric functions in the SecuBSP COM control (SecuBSPMxCOM.DLL) in a web application

Samples using SecuBSPMx.NET.DLL

- **Samples\C#** - Sample program written in C# that demonstrates the use of biometric functions in the SecuBSP .NET library (SecuBSPMx.NET.DLL)
- **Samples\VB.NET** - Sample program written in Visual Basic .NET that demonstrates the use of biometric functions in the SecuBSP .NET library (SecuBSPMx.NET.DLL)
- **Samples\bspdemo_advanced_cs\ExportAuditData** - Sample program written in C# that demonstrates the data auditing function in the SecuBSP .NET library (SecuBSPMx.NET.DLL).
- **Samples\bspdemo_advanced_cs\ FDxToSecuBSP** - Sample program written in C# that demonstrates the fingerprint data conversion function in the SecuBSP .NET library (SecuBSPMx.NET.DLL) (from FDx 400 byte minutiae format (SG400) to SecuBSP FIR format)
- **Samples\bspdemo_advanced_cs\ SecuBSPToFDx** - Sample program written in C# that demonstrates the fingerprint data conversion function in the SecuBSP .NET library (SecuBSPMx.NET.DLL) (from SecuBSP FIR to FDx 400 byte minutiae format (SG400))

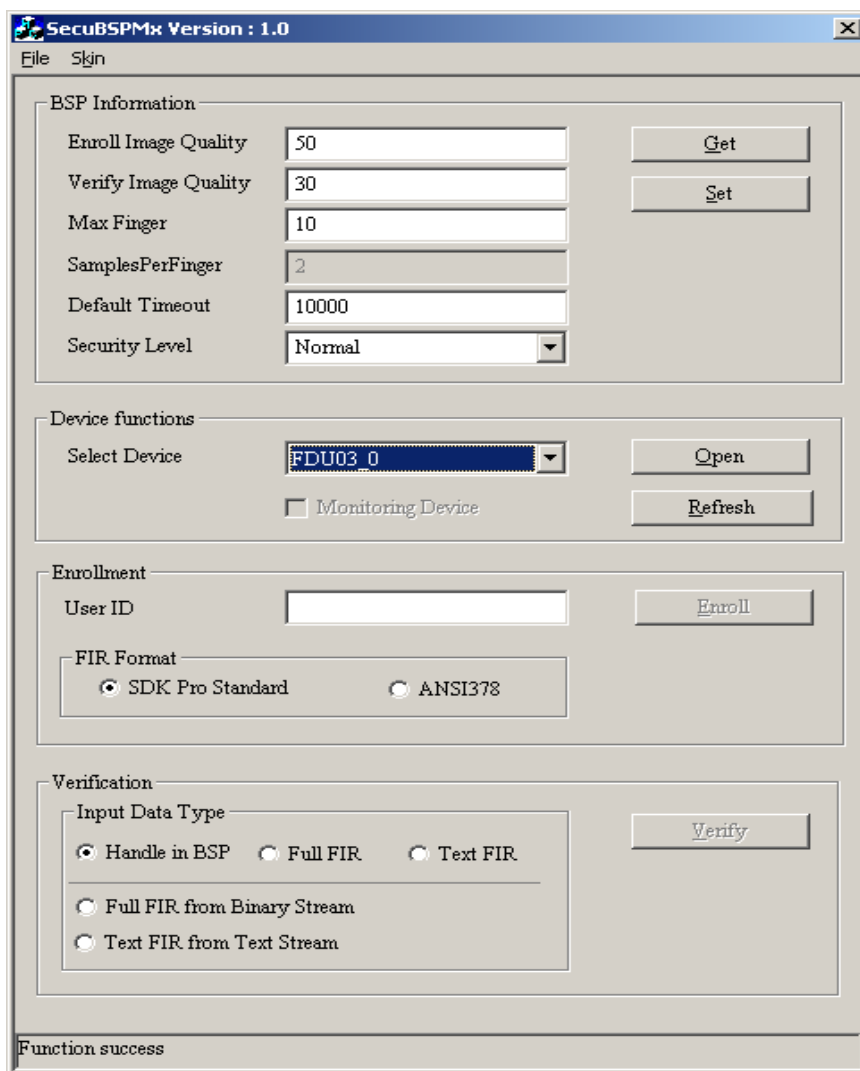
2.3.6. I.NET Framework 2.0 Redistributable

- **dotnetfx.exe:** .NET Framework Redistributable file (for 32-bit)
- **NetFx20SP1_x86:** .NET Framework Service Pack 1 (for 32-bit)
- **NetFx64.exe:** .NET Framework Redistributable file (for 64-bit)
- **NetFx20SP1_x64:** .NET Framework Service Pack 1 (for 64-bit)

2.4. Demo Program

After installing the SecuBSP SDK Pro program, the SecuBSP Demo Program icon will be added to the SecuBSP SDK Pro folder under *Programs*. This program demonstrates the various functions and features of the SecuBSP module using the sample source code.

Run the SecuBSP Demo program to test the basic functions of SecuBSP and verify a successful installation. Execute the SecuBSP Demo Program to display the following window:



The SecuBSP Demo Program includes four sections:

- BSP Information
- Device Functions
- Enroll
- Verify

2.4.1. BSP Information

You can set or get SecuBSP default values in this section. When you execute BSPdemo.exe, initially it displays BSP default values. Click **Get** to retrieve the default values. Otherwise enter values into the field(s) you wish to change, and click **Set** to apply the new settings.

- **Enroll Image Quality** - Quality level of the fingerprint image for enrollment; value range is 30 to 100
- **Verify Image Quality** - Quality level of the fingerprint image for verification; value range is 0 to 100
- **Max Finger** - Maximum number of fingers that can be enrolled into one template
- **SamplesPerFinger** - Number of fingerprint samples for enrollment; default value is 2. (Currently, it cannot be altered.)
- **Default Timeout** - Timeout value for enrollment (in milliseconds)
- **Security Level** - Security level settings range from 1 to 9. Default value is 5 (mid-level of security). The higher the security level is, the lower the FAR (False Acceptance Rate) becomes and the higher the FRR (False Rejection Rate) becomes.

The Basics of SecuBSP Security Levels

Administrators need to understand the basic concepts behind security levels. There is always a trade-off between user convenience and security, since increasing the security level will affect fingerprint verification. Increased security generally results in a higher rate of false rejections (false rejection rate, or FRR). False rejection occurs when an authorized (registered) user's fingerprint is not matched successfully to the stored sample. Likewise, decreasing the security level will result in an increased frequency of false acceptance (false acceptance rate, or FAR). FAR is potentially more serious, since this means an unauthorized user is granted access. The net result of FRR is usually nothing more than an inconvenience to users. More minutiae data (fingerprint characteristics) are required for a match when the security level and corresponding matching threshold is increased, resulting in a more discriminating fingerprint reader. In high security environments, users will need to be more attentive to physical technique when applying fingerprints, and may require basic education regarding environmental considerations such as bright light and moisture, and their possible effects on fingerprint capturing. If these steps are not taken, false rejections may occur. When this happens, the user must re-apply the fingerprint for another attempt at matching. In summary, where a higher false reject rate (FRR) can cause some inconvenience, the net results of a higher false acceptance rate (FAR) can be far more serious if critical resources are at stake.

SecuGen readers use *security levels* to fine-tune the fingerprint matching process, allowing different installations to emphasize security over convenience, or vice versa, depending on the requirements of a site. Security levels range from Lowest (1) to Highest (9). SecuBSP defaults to the "Normal" (5) security setting, designed for optimum speed and security for verification.

Note: FAR and FRR are directly and inversely proportional, so it is important to establish security policies based upon this fact. If a higher security level is selected, the FAR will be decreased, and the FRR will be increased. If a lower security level is selected, the FAR will be increased, and the FRR will be decreased

2.4.2. Device Functions

This section shows the list of fingerprint readers currently connected to the system. Any fingerprint reader that is connected can be selected for use in SecuBSP. After selecting the reader, click **Open** to activate the reader in SecuBSP. The Auto_Detect option will automatically search for the reader that was used during the last SecuBSP session.

2.4.3. Enroll

The “Enroll” function tests the user enrollment function. If you enter a User ID, it will be inserted into the FIR as a payload.

2.4.4. Verify

The “Verify” function tests the verification function. The BSP can perform verification in three ways: by using the FIR handle, the FIR itself, or text-encoded FIR. If the enrolled fingerprint data contains a payload (User ID in this example), the UserID will be returned after successful verification. For each type, the BSP Demo Program also shows how FIR can be converted into a data stream for file saving or network transmission.

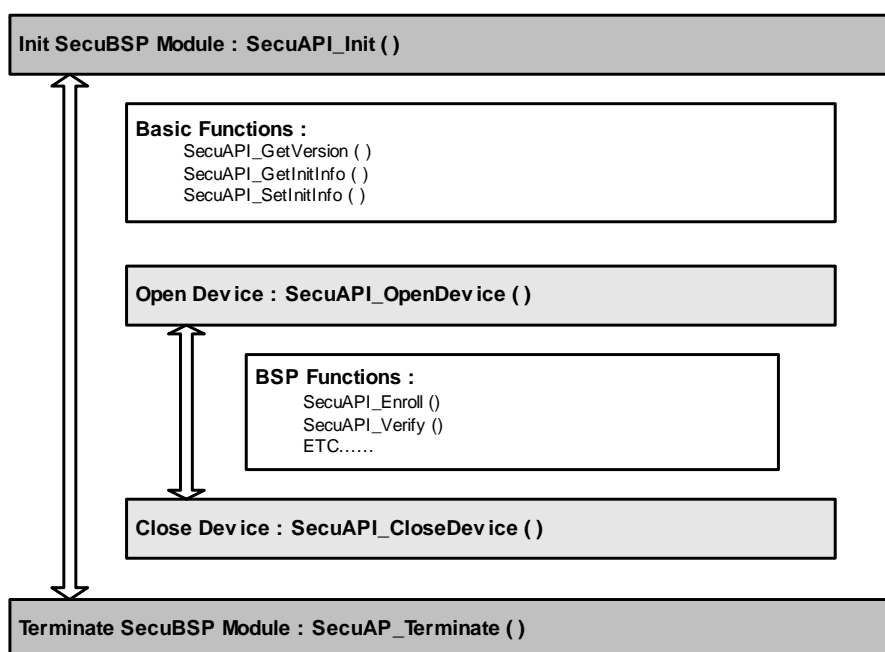
For further information, refer to [Chapter 3. SecuBSP Programming in C/C++](#).

Chapter 3. SecuBSP Programming in C/C++

This chapter explains programming in C/C++. Sample source code was written using SecuBSPMx.DLL.

The following diagram shows SecuBSP's function structure.

Function Structure



3.1. Check Module Validity

The SecuBSPMx.DLL is digitally signed to prevent tampering and to ensure the integrity of its contents. The SecuBSP SDK Pro provides a method for checking the SecuBSPMx.DLL module for signs of tampering. Although this validity check is optional, SecuGen recommends using this feature as a precaution.

The **SecuAPI_CheckValidity()** function is called to check module validation, and it uses "SecuBSPMx.DLL" as a parameter. The function prototype is defined in "SecuAPI_CheckValidity.h," and the library file is "SecuAPI_CheckValidity.lib."

SecuGen recommends using the function SecuAPI_CheckValidity() to ensure SecuBSP integrity.

```

#include "SecuAPI_CheckValidity.h"

bool IsValidModule()
{
    if (SecuAPI_CheckValidity(_T("SecuBSPMx.DLL")) == SecuAPIERROR_NONE)
        return true;
}
  
```

```
    else
        return false;
}
```

3.2. Module Initialization & Termination

The SecuBSP module must be initialized before use and is done using the **SecuAPI_Init()** function. **SecuAPI_Init()** returns the handle of the SecuBSP module, which is used for the duration of the session (i.e. for as long as the application needs to use the SecuBSP module).

The **SecuAPI_Terminate()** function must be called to terminate use of the SecuBSP Module. The **SecuAPI_Terminate()** function frees the memory allocated to the SecuBSP module and releases the handle.

3.2.1. Initialize Module

SecuAPI_Init() is the function that initializes the SecuBSP module. **SecuAPI_Init()** returns the Handle of the SecuBSP module used in the application.

```
SecuAPI_HANDLE  g_hBSP;                // SecuBSP module handle
...
// Initialize BSP Module
if ( SecuAPI_Init(&g_hBSP) != SecuAPIERROR_NONE )
{
    // Init module failed. Show error message.
}
// Init success.
```

3.2.2. Terminate Module

The **SecuAPI_Terminate()** function frees the memory used by the SecuBSP module and must be called prior to terminating the application.

```
SecuAPI_HANDLE  g_hBSP;                // SecuBSP module handle
SecuAPI_RETURN  ret;
. . .
ret = SecuAPI_Terminate(g_hBSP);       // Terminate BSP module
```

3.3. Device Functions

The functions that control the fingerprint reader can only be used after the reader is first opened. Before opening the reader, however, the **SecuAPI_EnumerateDevice()** function can be used to get information about the fingerprint recognition readers connected to the system. After calling **SecuAPI_EnumerateDevice()** to retrieve device information, open the reader by calling **SecuAPI_OpenDevice()**.

Device-related functions:

```
SecuAPI_EnumerateDevice()
SecuAPI_OpenDevice()
SecuAPI_CloseDevice()
SecuAPI_GetDeviceInfo()
SecuAPI_AdjustDevice()
```

3.3.1. Enumerating Devices

The **SecuAPI_EnumerateDevice()** function is used to retrieve specific device information: the number of readers installed, and the device IDs. The Device ID consists of the device name and an instance number. The lower byte of the Device ID represents the device name; the higher byte represents the device instance number. For example, if there are two FDP02 (parallel port) readers installed on the system, the device ID of one would be 0x0001, and the device ID of the other would be 0x0101. The maximum number of readers supported is 254 and is defined in **SecuAPI_MAX_DEVICE**.

```
SecuAPI_RETURN ret;
SecuAPI_UINT32 nDeviceNum;
SecuAPI_DEVICE_ID **pDeviceList;

// Get device list in the PC.
ret = SecuAPI_EnumerateDevice(g_hBSP, &nDeviceNum, pDeviceList);
CString device_name;
WORD instance_num;

for ( i = 0; i < nDeviceNum; i++)
{
    if (pDeviceList[i] & 0x00FF == SecuAPI_DEVICE_NAME_FDP02)
        device_name = "FDP02";
    else if (pDeviceList[i]&0x00FF == SecuAPI_DEVICE_NAME_FDU02)
        device_name = "FDU02";
}
```

SecuAPI_EnumerateDevice() returns the number of readers installed in the second parameter, **nDeviceNum**, and the device IDs in the third parameter, **pDeviceList**.

The SecuBSP module internally allocates the memory used by **pDeviceList**, so it is unnecessary for the application to allocate memory for **pDeviceList**.

Predefined device names in SecuAPI

Device Name	Value	Notes
SecuAPI_DEVICE_NAME_FDP02	0x01	FDP01 & FDP02 parallel devices
SecuAPI_DEVICE_NAME_FDU02	0x02	FDU02 USB devices
SecuAPI_DEVICE_NAME_FDU03	0x03	FDU03 & SDU03 USB devices
SecuAPI_DEVICE_NAME_FDU04	0x04	FDU04 USB devices

3.3.2. Device Initialization

SecuAPI_OpenDevice() is used to initialize the reader. The **SecuAPI_Enumerate()** function returns the list of device IDs that can be initialized using **SecuAPI_OpenDevice()**. The **SecuAPI_OpenDevice()** function will automatically open the reader installed on the system if **SecuAPI_DEVICE_ID_AUTO** is specified as the device ID.

```
// open device using option that auto_detect
m_DeviceID = SecuAPI_DEVICE_ID_AUTO;
ret = SecuAPI_OpenDevice(g_hBSP, m_DeviceID)// open device
if ( ret != SecuAPIERROR_NONE )           // check error // // fail to open device
else
// device open success.
```

3.3.3. Closing Device

Calling the function **SecuAPI_CloseDevice()** will close the reader currently being used by the SecuBSP Module.

```
// close device.
ret = SecuAPI_CloseDevice(g_hBSP, m_DeviceID);
if ( ret != SecuAPIERROR_NONE )
{
    // fail to close device
}
```

3.3.4. Getting Device Information

SecuAPI_GetDeviceInfo() is used to get detailed device information. The function takes the device ID for the second parameter, 0 for the third parameter, and the pointer **SecuAPI_DEVICE_INFO** for the fourth parameter.

```
// get device Info
SecuAPI_DEVICE_INFO device_info;
ret = SecuAPI_GetDeviceInfo(g_hBSP, m_DeviceID, 0, &device_info);
if ( ret != SecuAPIERROR_NONE )
{
    // close device if get device info is failed.
}
```

Currently, two data structures are supported: **SecuAPI_DEVICE_INFO_0** and **SecuAPI_DEVICE_INFO_1**. Using **SecuAPI_DEVICE_INFO_1**, you can retrieve the fingerprint reader serial number, firmware version and image resolution (DPI) in addition to the information obtained from **SecuAPI_DEVICE_INFO_0**. This function takes the value 1 for the third parameter.

```
SecuAPI_DEVICE_INFO_1 DeviceInfo1;
memset(&DeviceInfo1, 0, sizeof(DeviceInfo1));
DeviceInfo1.StructureType = 1;
SecuAPI_RETURN ret = SecuAPI_GetDeviceInfo(g_hSecuAPI, m_DeviceID, 1, &DeviceInfo1);

if (SetResultValue(ret))
{
    m_nDeviceBrightness = DeviceInfo1.Brightness;
```

```

    m_nDeviceContrast = DeviceInfol.Contrast;
    m_nDeviceGain = DeviceInfol.Gain;
    m_nDeviceHeight = DeviceInfol.ImageHeight;
    m_nDeviceWidth = DeviceInfol.ImageWidth;
    m_szDeviceSN = CString(DeviceInfol.DevicesSN);
    m_nDeviceImageDPI = DeviceInfol.ImageDPI;
    m_sDeviceFWVersion.Format("%x", DeviceInfol.FWVersion);
}

```

3.3.5. Using Auto-On

Auto-On is a function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. To use this function, Auto-On should be enabled using **SecuAPI_MonitorDevice()**. Once Auto-On is enabled, the application can receive a message from the device driver whenever an Auto-On event occurs in the device.

When calling **SecuAPI_MonitorDevice()**, pass the handle of the window which will receive the Auto-On message. The Auto-On message is defined as **SecuAPI_WM_DEVICE_EVENT** (0x81000).

Note: Auto-On is not supported by FDP02 or FDU02-based readers.

- **Enabling Auto-On**

```
SecuAPI_MonitorDevice(m_hBSP, m_DeviceID, true, m_hWnd);
```

- **Disabling Auto-On**

```
SecuAPI_MonitorDevice(m_hBSP, m_DeviceID, false, m_hWnd);
```

- **Handling an Auto-On Event**

```

BEGIN_MESSAGE_MAP(CBSPDemoDlg, CDialog)
    ON_MESSAGE (SecuAPI_WM_DEVICE_EVENT, OnDeviceEvent) // Auto-On/off message
END_MESSAGE_MAP()

LRESULT CBSPDemoDlg::OnDeviceEvent(WPARAM wParam, LPARAM lParam)
{
    // Do whatever you want
    WORD isfinger = wParam;

    if(wParam == SecuAPI_DEVICE_FINGER_ON)
    {
        m_StatusBar = TEXT("Monitoring Device - Finger On");
        MessageBeep(-1);
    }
    else if ( wParam == SecuAPI_DEVICE_FINGER_OFF)
    {
        m_StatusBar = TEXT("Monitoring Device - Finger Off");
    }

    UpdateData(FALSE);

    return 1;
}

```

3.4. Fingerprint Enrollment

Fingerprint data processed in the SecuBSP are in Fingerprint Identification Record (FIR) format. Templates are FIR data used for enrollment.

The **SecuAPI_Enroll()** function is used to enroll (register) fingerprint data.

```
ret = SecuAPI_Enroll(
    g_hBSP,                // SecuBSP Handle
    NULL,
    &g_hEnrolledFIR,        // handle of FIR to be enrolled
    NULL,                  // input payload
    -1,                    // capture timeout (ms)
    NULL,                  // audit data
    NULL                   // Windows options
);
. . .
SecuAPI_FreeFIRHandle(g_hBSP, g_hEnrolledFIR);
```

The FIR returned by calling the **SecuAPI_Enroll()** function resides in the SecuBSP module. The application cannot know the FIR structure, but instead refers to it only as a handle.

3.4.1. Retrieving the FIR

The FIR handle returned by the **SecuAPI_Enroll()** function cannot be used directly in a file or a database unless it is first retrieved from SecuBSP.

SecuAPI_GetFIRFromHandle() or **SecuAPI_GetExtendedFIRFromHandle()** is used to retrieve the FIR associated with a FIR handle. The following example shows how to retrieve the FIR from a FIR handle. **SecuAPI_GetFIRFromHandle()** returns FIR in default format: SecuAPI_FIR_FORMAT_STANDARDPRO. To get the FIR in ANS I378 format, use SecuAPI_GetExtendedFIRFromHandle().

For more information on how to get the FIR in ANSI378 format, refer to [Section 3.11 ANSI378 Format Fingerprint Data](#).

```
SecuAPI_FIR g_FIR;
ret = SecuAPI_GetFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_FIR);
```

The FIR consists of Format, Header, and Data fields. The Data field contains the address of the contiguous fingerprint data area. The fingerprint data is of variable length, thus the Data Length field in the header contains the size of the fingerprint data. The total size of the FIR is the sum of the format size, the header size, and the fingerprint data size.

$$\text{FIR length} = \text{Format length} + \text{Header length} + \text{Fingerprint data length}$$

In order to use FIR in a file, database, or network transmission, the FIR must be converted into stream form. (This will be explained further in the following section.)

SecuAPI_FIR Structure

FIR Field		Description
Format		FIR Format
Header	Length	FIR Header length
	DataLength	Fingerprint Data Length
	Version	FIR Version
	DataType	Type of FIR
	Purpose	Purpose of FIR
	Quality	Currently not used
	Reserved	Reserved area
Data		The address of contiguous fingerprint data

3.4.2 Converting the FIR into a Binary Stream

The FIR must be converted into a stream format in order to be used in a file or a database. The example below shows the process for converting the FIR into a stream. When the FIR is no longer needed, free the memory space allocated for the FIR by using the **SecuAPI_FreeFIR()** function. **SecuAPI_FreeFIR()** frees the memory allocated for the fingerprint data inside the SecuBSP module.

```

SecuAPI_FIR          g_FIR;
DWORD               length;

// get FIR from FIR handle
ret = SecuAPI_GetFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_FIR);

if ( ret == SecuAPIERROR_NONE )
{
    // make stream data from FIR
    BYTE* binary_stream;
    length = sizeof(g_FIR.Format) + g_FIR.Header.Length + g_FIR.Header.DataLength;
    binary_stream = new BYTE[length];
    memcpy(&binary_stream[0], &g_FIR.Format,
        sizeof(g_FIR.Format));
    memcpy(&binary_stream[sizeof(g_FIR.Format)],
        &g_FIR.Header, g_FIR.Header.Length);
    memcpy(&binary_stream[sizeof(g_FIR.Format)+
        g_FIR.Header.Length], &g_FIR.Data, g_FIR.Header.DataLength);

    //save binary data to file or database
    delete [ ] binary_stream;
}

. . .

SecuAPI_FreeFIR(g_hBSP, &g_FIR);    // free FIR

```

3.4.3. Retrieving the Text-Encoded FIR

In some programming environments such as Visual Basic or Delphi, or in a web environment, the text-encoded FIR may be needed.

The **SecuAPI_GetTextFIRFromHandle()** function retrieves the text-encoded FIR from SecuBSP. Unlike the **SecuAPI_GetFIRFromHandle()** function, the FIR that is returned is text data with a format different from the

standard C structure of a normal FIR. When calling **SecuAPI_GetTextFIRFromHandle()**, it is necessary to specify whether or not to get the text data in a multi-byte format. The **SecuAPI_GetTextFIRFromHandle()** function takes **SecuAPI_TRUE** as the fourth parameter to get the multi-byte text data, otherwise, **SecuAPI_FALSE**.

```
SecuAPI_FIR_TEXTENCODE      g_firText;

// get Text FIR from FIR handle
ret = SecuAPI_GetTextFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_firText, SecuAPI_FALSE);

if ( ret == SecuAPIERROR_NONE )
{
    char* text_stream;
    DWORD length;
    length = lstrlen(g_firText.TextFIR);
    if (g_firText.IsWideChar == SecuAPI_TRUE)
        text_stream = new char [(length + 1)*2];
    else
        text_stream = new char [length + 1];
    memcpy(text_stream &g_firText.TextFIR, length);

    // save text_stream to File or Database
    delete [] text_stream
}

SecuAPI_FreeTextFIR(g_hBSP, &g_firText); // Free TextFIR handle
```

The parameter **g_firText** returned by calling **SecuAPI_GetTextFIRFromHandle()** contains the information needed to determine whether the text data is in multi-byte format or not, and the address of the text-encoded FIR.

SecuAPI_FIR_TEXTENCODE Structure

Member	Description
IsWideChar	Specifies whether it is multi-byte format or not (SecuAPI_TRUE : Multi-byte)
TextFIR	The address of the text-encoded FIR

3.5. Verification

The **SecuAPI_Verify ()** function captures the fingerprint data from a reader and compares that sample against the previously enrolled FIR. This function returns the verification result in the third parameter as “True” or “False” after comparison.

As explained in section 3.4, the FIR may be in any of three forms: the FIR Handle, the FIR itself, or the text-encoded FIR. Each form of the FIR can be specified in the Form member of the **SecuAPI_INPUT_FIR** structure.

SecuAPI_INPUT_FIR Structure

Member		Description
Form	SecuAPI_FIR_FORM_HANDLE	FIR handle form
	SecuAPI_FIR_FORM_FULLFIR	FIR form
	SecuAPI_FIR_FORM_TEXTENCODE	Text FIR form
InputFIR	FIRinBSP	FIR Handle
	FIR	Pointer of FIR Handle
	TextFIR	Pointer of Text FIR

3.5.1. Verification with the FIR Handle

In order to verify with the FIR Handle, input **SecuAPI_FIR_FORM_HANDLE** in the form of **SecuAPI_INPUT_FIR**, input the FIR Handle in the **InputFIR.FIRinBSP** member, then call the **SecuAPI_Verify ()** function.

```
SecuAPI_BOOL result;           // verification result
SecuAPI_INPUT_FIR inputFIR;   // set input FIR.

inputFIR.Form = SecuAPI_FIR_FORM_HANDLE; inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR;

if ( g_hBSP != SecuAPI_INVALID_HANDLE )
ret = SecuAPI_Verify(
    g_hBSP,                    // handle of SecuBSP module
    &inputFIR,                  // stored FIR
    &result,                    // result of verification
    NULL,                      // payload
    10000,                     // capture timeout
    NULL,                      // audit data
    NULL                       // window option
);

if ( result == SecuAPI_TRUE )
    // Verification success
else
    // verification failed
```

3.5.2. Verification with the FIR

In order to verify using the FIR, input **SecuAPI_FIR_FORM_FULLFIR** in the Form of **SecuAPI_INPUT_FIR**, input the address of the FIR in the **InputFIR.FIR** member, and then call the **SecuAPI_Verify ()** function.

```
SecuAPI_INPUT_FIR inputFIR;   // set input FIR.
```

```

SecuAPI_BOOL result;           // verification result
SecuAPI_INPUT_FIR inputFIR;

// Set FIR Form to Full FIR
inputFIR.Form = SecuAPI_FIR_FORM_FULLFIR;
inputFIR.InputFIR.FIR = &g_FIR;

// Matching
if ( g_hBSP != SecuAPI_INVALID_HANDLE)//check SecuBSP handle
{
    ret = SecuAPI_Verify(
        g_hBSP,                // handle of SecuBSP module
        &inputFIR,              // stored FIR
        &result,               // result of verification
        NULL,                  // payload in FIR
        5000,                  // timeout for scan image
        NULL,                   // audit data
        NULL);                 // window option
};

if ( result == SecuAPI_TRUE)
    // verification success
else
    // verification failed

```

If the fingerprint data is in a file or database, the stream must first be converted into the FIR format. The code below shows the conversion process.

```

// convert binary stream data to FIR
SecuAPI_FIR g_FIR;
char * binary_stream;

// filll binary stream from file or DB
total_length = strlen(binary_stream);
format_length = sizeof(g_FIR.Format);
header_length = sizeof(g_FIR.Header);
data_length = strlen(binary_stream)-(format_length + data_length);

memcpy(&g_FIR.Format, &binary_stream[0], format_length + header_length);
memcpy(&g_FIR.Data, &binary_stream[format_length+header_length], data_length);

```

3.5.3. Verification with Text-Encoded FIR

If the FIR is in text-encoded format, use **SecuAPI_FIR_FORM_TEXTENCODE** for the form of **SecuAPI_INPUT_FIR**, and the address of text-encoded FIR in the **InputFIR.TextFIR** member when calling the **SecuAPI_Verify()** API. The following example shows the verification process using the text-encoded FIR.

```

SecuAPI_INPUT_FIR inputFIR;           //set input FIR.
inputFIR.Form = SecuAPI_FIR_FORM_TEXTENCODE; //set input FIR to text encoded FIR
inputFIR.InputFIR.TextFIR = &g_TextFIR;

if ( g_hBSP != SecuAPI_INVALID_HANDLE) //check SecuBSP handle
{
    ret = SecuAPI_Verify(
        g_hBSP,                    // handle of SecuBSP module

```

```

    &inputFIR,                // stored FIR
    &result,                  // result of verification
    NULL,                     // payload
    5000,                     // capture timeout
    NULL,                     // audit data
    NULL);                    // window option

```

If the fingerprint data is in a file or database, the stream needs to be converted into the text-encoded FIR (**SecuAPI_FIR_TEXTENCODE**).

```

SecuAPI_FIR_TEXTENCODE  g_TextFIR;           // set input FIR.
char* text_stream;
// fill text_stream buffer from file or database
length = strlen(text_stream);

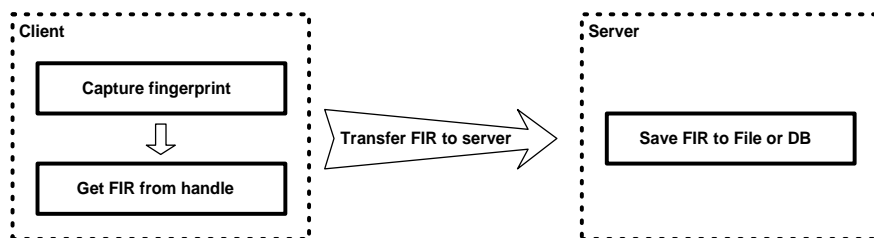
// fill g_TextFIR structure
g_TextFIR.IsWideChar = SecuAPI_TRUE;         // depends on application
memcpy(&g_FIR.TextFIR, text_stream, length);

```

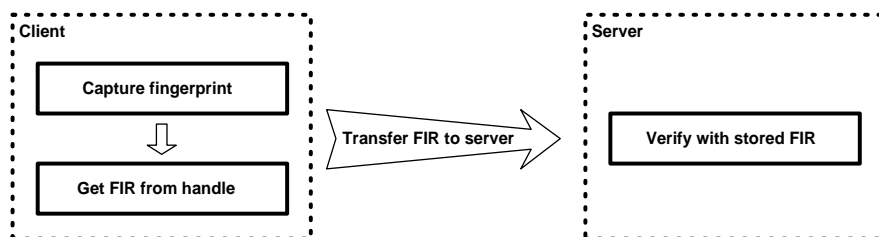
3.6. The Client/Server Environment

In a client/server environment, the fingerprint capture function is executed on the client system while the matching and storing of the fingerprint data is carried out on the server system. For this reason, high-level functions such as **SecuAPI_Enroll()** and **SecuAPI_Verify()** cannot be used. Instead, low level APIs such as **SecuAPI_Capture()**, **SecuAPI_CreateTemplate()**, and **SecuAPI_VerifyMatch()** should be used.

Fingerprint Enrollment Process in a Client/Server environment



Fingerprint Verification Process in a Client/Server Environment



3.6.1. Capturing Fingerprint Data

To capture fingerprint data on a client system, use the **SecuAPI_Capture()** function with the purpose of capturing in the second parameter.

```
SecuAPI_RETURN ret;
SecuAPI_FIR_PURPOSE m_Purpose = SecuAPI_FIR_PURPOSE_VERIFY;
ret = SecuAPI_Capture(
    g_hBSP,                // handle of SecuBSP Module
    m_Purpose,               // purpose of capture
    &g_hCapturedFIR,      // handle of captured FIR
    10000,                // capture timeout
    NULL,                 // audit data
    NULL                   // window option
);
```

The value that is specified for **SecuAPI_FIR_PURPOSE** will determine which wizard is displayed.

SecuAPI_FIR_PURPOSE Values

Value	Description
SecuAPI_FIR_PURPOSE_VERIFY	For Verification
SecuAPI_FIR_PURPOSE_IDENTIFY	For Identification (currently not used)
SecuAPI_FIR_PURPOSE_ENROLL	For Registration
SecuAPI_FIR_PURPOSE_AUDIT	For Auditing (currently not used)
SecuAPI_FIR_PURPOSE_UPDATE	For Updating (currently not used)

For example, if the purpose is **SecuAPI_FIR_PURPOSE_VERIFY**, the verification wizard window will be displayed on the client system to users.

After the fingerprint capture process is completed, the FIR or Text-encoded FIR can be retrieved from the FIR Handle using **SecuAPI_GetFIRFromHandle()** or **SecuAPI_GetTextFIRFromHandle()**. To transmit the fingerprint data on a network, the FIR data should be converted to a binary stream form.

3.6.2. Verification on a Server System

While the **SecuAPI_Verify()** function is used to match the live fingerprint data against the previously enrolled FIR data, the **SecuAPI_VerifyMatch()** is used to match the FIR data transmitted on a network against the previously enrolled FIR data. Therefore, **SecuAPI_VerifyMatch()** function takes two FIRs, the transmitted FIR data and the enrolled FIR data.

```
SecuAPI_RETURN ret;
SecuAPI_INPUT_FIR storedFIR, inputFIR;
SecuAPI_FIR fir1, fir2;
SecuAPI_BOOL result;

// Read stored data and convert into FIR(fir1)

storedFIR.Form = SecuAPI_FIR_FORM_FULLFIR; // stored FIR
storedFIR.InputFIR.FIR = &fir1;

// Read input stream and convert into FIR(fir2)
```

```

// input FIR to be compared
inputFIR.Form = SecuAPI_FIR_FORM_FULLFIR;
inputFIR.InputFIR.FIR = &fir2;

ret = SecuAPI_VerifyMatch(
    g_hBSP,                // handle of SecuBSP module
    &inputFIR,              // input FIR for matching
    &storedFIR,             // stored FIR
    &result,               // result of matching
    &payload               // payload
);

if (result == SecuAPI_TRUE)
    // verification success

```

3.7. Payload

No two fingerprint samples from a user are likely to be identical. For this reason, it is not possible to directly use the fingerprint samples as cryptographic keys. SecuBSP does, however, allow a template to be closely bound to a cryptographic key. In the enrollment process, the fingerprint data can contain a cryptographic key, and when the verification is successfully done, the key will be released. The cryptographic key that is stored within the FIR is called Payload. Payload can also be any other information such as a password or a user name.

The Payload can be retrieved after successful fingerprint verification using the **SecuAPI_Verify()** or **SecuAPI_VerifyMatch()** function. With this function, any data can be transmitted securely.

SecuAPI_FIR_PAYLOAD Structure

Member	Description
Length	Length of payload
Data	Data of payload

3.7.1. Inserting Payload

There are two ways to insert Payload into the FIR: using **SecuAPI_Enroll()** or **SecuAPI_CreateTemplate ()**. To insert Payload when creating the FIR by using **SecuAPI_Enroll()**, populate the **SecuAPI_FIR_PAYLOAD** structure. The first member is the length of the payload, and the second member is the actual data.

```

CString temp = "Test Payload";    // payload
SecuAPI_FIR_PAYLOAD payload;

payload.Length = strlen(temp) + 1; // fill payload structure
payload.Data = new SecuAPI_UINT8 [payload.Length];
memcpy(payload.Data, temp, payload.Length);

ret = SecuAPI_Enroll(
    g_hBSP,                // SecuBSP handle
    NULL,
    &g_hEnrolledFIR,        // handle of enrolled FIR
    &payload,               // input payload
    5000,                  // capture timeout (ms)
    NULL,
    NULL                   // windows options.

```

```

);

CString m_szPayload = "Test Payload"; //Set payload
SecuAPI_INPUT_FIR    inputFIR;        // input FIR
SecuAPI_FIR_PAYLOAD  payload;

//set type of input FIR
inputFIR.Form = SecuAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR;

// copy original FIR to input FIR
payload.Length = lstrlen(m_szPayload) + 1;
payload.Data = new SecuAPI_UINT8 [payload.Length];
memcpy(payload.Data, m_szPayload, payload.Length);

// Create new template that includes payload
ret = SecuAPI_CreateTemplate(g_hBSP, &inputFIR, NULL, &g_hNewTemplate, &payload);

delete[] payload.Data;

SecuAPI_FreeFIRHandle(g_hBSP, g_hEnrolledFIR);

// Delete original FIR
g_hEnrolledFIR = g_hNewTemplate;

// Overwrite original FIR with new FIR
g_hNewTemplate = 0;

```

3.7.2. Retrieving Payload from the Template

Both the **SecuAPI_Verify()** and the **SecuAPI_VerifyMatch()** functions return the Payload when (1) the FIR data contains payload data, and (2) the verification is successful.

```

SecuAPI_FIR_PAYLOAD payload;
if ( g_hBSP != SecuAPI_INVALID_HANDLE ) // check SecuBSP handle
{
    ret = SecuAPI_Verify(
        g_hBSP,                // handle of SecuBSP module
        &inputFIR,              // stored FIR
        &result,                // result of verification
        &payload,               // payload in FIR
        10000,                 // timeout
        NULL,                   // audit data
        NULL                     // window option
    );
}

if ( result == SecuAPI_TRUE )
{
    CString msg;
    msg.Format("Verified !!! (Payload : %s)", (LPTSTR)payload.Data);
    MessageBox(msg);
}
else
{
    // verification failed
}

```

```
}

```

3.8. Load Resource File

External resource DLLs can be loaded for additional language support. SecuBSP SDK Pro provides both English (default) and Korean resource files. **Note:** Documentation for UI customization is provided separately.

```
SecuAPI_SetSkinResource( "SBSP2Kor.dll" );
```

3.9. Data Conversion Functions

3.9.1. Importing FDx Data

To import 400 byte fingerprint minutiae formatted data (from FDx SDK) and convert into the SecuBSP FIR format, use **SecuAPI_FDxToSecuBSP()**. The FDx data import sample program can be found in Samples\AdvancedPgms\FDxToSecuBSP in the SecuBSP SDK Pro directory.

```
BYTE                m_minData[400];
SecuAPI_FIR_HANDLE  m_hFIR;

err = SecuAPI_FDxToSecuBSP(m_hSecuBSP, m_minData, 400, MINCONV_TYPE_FDU,
SecuAPI_FIR_PURPOSE_VERIFY, &m_hFIR) )
```

3.9.2. Exporting FDx Data

To export SecuBSP FIR format data and convert into 400 byte fingerprint minutiae formatted data (to use with FDx SDK), use **SecuAPI_SecuBSPToFDx()**. The exported FDx data is contained in the SecuAPI_EXPORT_DATA structure. To use **SecuAPI_SecuBSPToFDx()**, include the SecuAPI_Export.h header file. For more information about SecuAPI_EXPORT_DATA, refer to the *SecuAPI Reference Manual*. The FDx data export sample program can be found in Samples\AdvancedPgms\SecuBSPToFDx in the SecuBSP SDK Pro directory.

```
SecuAPI_INPUT_FIR inputFIR;
inputFIR.Form = SecuAPI_FIR_FORM_FULLFIR;
inputFIR.InputFIR.FIR = &m_FullFIR;

if ( SecuAPIERROR_NONE != SecuAPI_SecuBSPToFDx(m_hSecuBSP, &inputFIR, &m_ExportData,
MINCONV_TYPE_FDU) )
    return;

// Display exported FIR data info
int i;

m_cszFingerNum.Format("%d", m_ExportData.FingerNum);
m_cszSamplesPerFinger.Format("%d", m_ExportData.SamplesPerFinger);

for ( i=0; i<11; i++ )
    pFPSelect[i]->EnableWindow(FALSE);

for ( i=0; i<m_ExportData.FingerNum; i++ )
{
```



```

        pFPSelect[m_ExportData.FingerData[i].FingerID]->EnableWindow(TRUE);
    }

    if ( m_ExportData.FingerNum > 0 )
        pFPSelect[m_ExportData.FingerData[0].FingerID]->SetCheck(1);

```

3.10. Audit Data

During the enrollment or verification process, audit data may be obtained for later use. Audit data contains the fingerprint image for each enrolled fingerprint. Image data may be extracted from audit data using **SecuAPI_SecuBSPToImage()**. The extracted image data is contained in the SecuAPI_EXPORT_AUDIT_DATA structure. For more information about SecuAPI_EXPORT_AUDIT_DATA, refer to the *SecuAPI Reference Manual*.

Note: To use **SecuAPI_ExportImage()**, include SecuAPI_Export.h. The audit data export sample program can be found in Samples\AdvancedPgms\ ExportImage in the SecuBSP SDK Pro directory.

```

SecuAPI_HANDLE  g_hSecuBSP = 0;
SecuAPI_FIR_HANDLE g_hAuditData = 0; // Will be used for exporting image data
SecuAPI_FIR_HANDLE g_hEnrolledFIR = 0;
SecuAPI_EXPORT_AUDIT_DATA* g_ExportImageData = 0;

// Get Audit Data
SecuAPI_RETURN err = SecuAPI_Enroll(g_hSecuBSP, NULL, &g_hEnrolledFIR, NULL, -1,
&g_hAuditData, NULL);

// Export image data from Audit data
SecuAPI_INPUT_FIR input;
input.Form = SecuAPI_FIR_FORM_HANDLE;
input.InputFIR.FIRinBSP = &g_hAuditData;

DWORD err = SecuAPI_SecuBSPToImage(g_hSecuBSP, &input, g_ExportImageData);

// Draw Image
if (err == 0)
{
    m_cwNumOfFingers.Format("Number of Fingers: %d", g_ExportImageData->FingerNum);
    m_cwSamplesPerFinger.Format("Samples Per Finger: %d", g_ExportImageData-
>SamplesPerFinger);
    m_cwImageWidth.Format("Image Width: %d", g_ExportImageData->ImageWidth);
    m_cwImageHeight.Format("Image Height: %d", g_ExportImageData->ImageHeight);
    UpdateData(FALSE);

    for (int i = 0; i < g_ExportImageData->FingerNum; i++)
    {
        BYTE fingerID = g_ExportImageData->AuditData[i].FingerID;
        m_btnFingerIDs[fingerID]->EnableWindow(TRUE);
    }
    m_btnFingerIDs[g_ExportImageData->AuditData[0].FingerID]->SetCheck(TRUE);
}

```

3.11. ANSI378 Format Fingerprint Data

SecuBSP SDK *Pro* uses extraction and matching algorithms that support the pure (unencrypted) ANSI-INCITS 378-2004 finger minutiae format standard for fingerprint templates, referred to as “ANSI378” in this document and elsewhere. However, by default, all fingerprint data generated by SecuBSP *Pro* is in SecuGen’s standard format that is encrypted with a 128-bit encryption algorithm.

SecuAPI FIR FORMAT

Format name	Description	Value
SecuAPI_FIR_FORMAT_STANDARDPRO	Encrypted FIR	3
SecuAPI_FIR_FORMAT_ANSI378	Pure ANSI378 Format	4

3.11.1. Retrieving ANSI378 format FIR data

SecuBSP *Pro* supports two formats: **SecuAPI_FIR_FORMAT_STANDARDPRO** and **SecuAPI_FIR_FORMAT_ANSI378**. When `SecuAPI_GetFIRFromHandle()` or `SecuAPI_GetTextFIRFromHandle()` is used, by default it returns `SecuAPI_FIR_FORMAT_STANDARDPRO`.

To get ANSI378 format FIR data, use **SecuAPI_GetExtendedFIRFromhandle()**.

```
SecuAPI_FIR g_FIR;
ret = SecuAPI_GetExtendedFIRFromHandle (g_hBSP, g_hEnrolledFIR, &g_FIR.
SecuAPI_FIR_FORMAT_ANSI378);
```

To get text type ANSI378 format data, use **SecuAPI_GetExtendedTextFIRFromHandle ()**.

```
SecuAPI_FIR_TEXTENCODE g_firText;

// get Text FIR from FIR handle
ret = SecuAPI_SecuAPI_GetExtendedTextFIRFromHandle (g_hBSP, g_hEnrolledFIR, &g_firText,
SecuAPI_FALSE, SecuAPI_FIR_FORMAT_ANSI378);
```

3.11.2. Restrictions on Using ANSI378 Format

SecuBSP *Pro* can generate ANSI378 format FIR data using **SecuAPI_GetExtendedFIRFromHandle()** or **SecuAPI_GetExtendedTextFIRFromHandle ()**. Note: the ANSI378 format FIR cannot store payload data because it is an open format.

In this version (SecuBSPMx.dll version 1.0), some APIs cannot accept ANSI378 FIR as an input parameter. If ANSI378 FIR is input in an unsupported function, it will return **SecuAPI_ERROR_INVALID_TYPE**.

The following functions can accept ANSI378 format FIR as an input parameter:

- SecuAPI_Verify()
- SecuAPI_VerifyMatch()

The following functions cannot accept ANSI378 format FIR as an input parameter:

- SecuAPI_Process()
- SecuAPI_CreateTemplate()
- SecuAPI_Enroll()
- SecuAPI_SecuBSPToFDx
- SecuAPI_SecuBSPToImage

Chapter 4. SecuBSP COM Programming

Developers using Microsoft Visual Basic or Borland Delphi or similar development environments will require an ActiveX component or COM module to simplify the development process. For this reason, SecuBSP SDK Pro also provides the SecuBSP COM module (SecuBSPMxCOM.DLL). This component is designed for web developers and for those using RAD such as Visual Basic or Delphi.

The SecuBSP COM module uses SecuBSPMx.DLL, but it does not support all SecuBSP functions. Fingerprint data type is offered only as a text type.

Required Components: SecuBSPMx.DLL and SecuBSPMxCOM.DLL

4.1. Module Initialization and Closure

4.1.1. Initializing the Module

[Visual Basic]

There are two ways to initialize the **SecuBSP COM** module: declare a new object or use the **CreateObject()** function. Using either method will have the same result.

Method 1: Declare new object

```
Dim objSecuBSP As SecuBSPCOMLib.APIInterface
...
Set objSecuBSP = New SecuBSPCOMLib.APIInterface
...
```

* To use this method, you must add the "SecuGen SecuBSP SDK Add-on Pack" in Reference of the Project menu.

Method 2: Use CreateObject()

```
Dim objSecuBSP As SecuBSPCOMLib.APIInterface
...
Set objSecuBSP = CreateObject("SecuBSPCOM.APIInterface")
...
```

[Delphi]

Use the **CreateObject()** function to initialize the **SecuBSP COM** module.

```
objSecuBsp : variant; // Declaration variable for SecuBSP Object
...
objSecuBsp := CreateOleObject('SecuBSPCOM.APIInterface');
```

4.1.2. Terminate the module use

Declare the object free when closing the application. Note that this is done automatically in Visual Basic when the application is closed.

[Visual Basic]

```
Set objSecuBSP = nothing ' Free SecuBSPCOM object
```

```
[Delphi]
objSecuBSP :=null;           // Free SecuBSPCOM object
```

4.2. Device Related Programming

The reader must be opened before it can be used. Use the **EnumerateDevice** method to determine which reader is linked to the system.

4.2.1. Listing Devices

Before opening the reader, use the **EnumerateDevice** method to determine the number and type of readers linked to the PC. Once this is activated, the number of readers linked to the PC will appear in the **DeviceNum** property, and the ID for each reader will appear in the **DeviceID** property. **DeviceID** is a LONG value array. **DeviceID** is composed of the device names and their instance numbers. **Note:** SecuBSP sets default settings for **FDP01**- and **FDP02**-based (parallel) fingerprint readers at **0x01**, **FDU02**-based (USB) fingerprint readers at **0x02**, **FDU03**- and **SDU03**-based (USB) fingerprint readers at **0x03**, and **FDU04**-based (USB) fingerprint readers at **0x04**.

DeviceID = Instance Number + Device Name

If there is only one reader for each type in the system, the instance number will be '0.' In this way, the device name has the same value as the device ID.

Predefined Device Names

Device Name	Value	Notes
SecuBSP_DEVICE_NAME_FDP02	1(0x01)	FDP01 & FDP02 parallel readers
SecuBSP_DEVICE_NAME_FDU02	2(0x02)	FDU02 USB readers
SecuBSP_DEVICE_NAME_FDU03	3(0x03)	FDU03 & SDU03 USB readers
SecuBSP_DEVICE_NAME_FDU04	4(0x04)	FDU04 USB readers

Predefined Device IDs

Device ID	Value	Notes
SecuBSP_DEVICE_ID_NONE	0(0x0000)	No devices
SecuBSP_DEVICE_ID_FDP02_0	1(0x0001)	The first instance of FDP01 or FDP02
SecuBSP_DEVICE_ID_FDU02_0	2(0x0002)	The first instance of FDU02
SecuBSP_DEVICE_ID_FDU03_0	3(0x0003)	The first instance of FDU03 or SDU03
SecuBSP_DEVICE_ID_FDU04_0	4(0x0004)	The first instance of FDU04
SecuBSP_DEVICE_ID_AUTO_DETECT	255(0x00FF)	Detect device automatically

The following is an example of how to use the **EnumerateDevice** method.

```
[Visual Basic]
objSecuBSP.EnumerateDevice // enumerate devices
...
For i = 0 To objSecuBSP.DeviceNum - 1
    Select Case objSecuBSP.DeviceID(i)
        Case SecuBSP_DEVICE_ID_FDP02_0
            'Parallel type device (FDP01 or FDP02)
        Case SecuBSP_DEVICE_ID_FDU02_0
            'USB type device (FDU02)
        Case SecuBSP_DEVICE_ID_FDU03_0
```

```

        'USB type device (FDU03 or SDU03)
    Case SecuBSP_DEVICE_ID_FDU04_0
        'USB type device (FDU04)

    End Select
Next i
...

[Delphi]

objSecuBSP.EnumerateDevice;
...
for i := 0 To objSecuBSP.DeviceNum - 1 do
begin
    Case objSecuBSP.DeviceID[i] of
        SecuBSP_DEVICE_ID_FDP02_0: // Parallel type device (FDP01 or FDP02)
        SecuBSP_DEVICE_ID_FDU02_0: // USB type device (FDU02)
        SecuBSP_DEVICE_ID_FDU03_0: // USB type device (FDU03 or SDU03)
        SecuBSP_DEVICE_ID_FDU04_0: // USB type device (FDU04)
    End;
end;

```

DeviceID is the number of devices found in the **DeviceNumber** property of the DeviceID (DeviceNumber). For example, DeviceID(0) will show the DeviceID of the first device.

4.2.2. Initializing the Device

The **OpenDevice** method is used to initialize the reader for **SecuBSP COM**. The reader must be initialized using the **OpenDevice** method before the device related functions, such as enrolling, verifying, and capturing, can work properly.

In the event that you are unsure which readers have been installed, use the **Enumerate** method to determine which readers have previously been installed.

```

[Visual Basic]
DeviceID = SecuBSP_DEVICE_ID_FDP02_0 //first instance of FDP02
objSecuBSP.OpenDevice(DeviceID)
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Open device success ...
Else
    ' Open device failed ...
End If

```

```

[Delphi]
DeviceID := SecuBSP_DEVICE_ID_PARALLEL;
objSecuBSP.OpenDevice(DeviceID);
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    //Open device success ...
Else
    // Open device failed ...
end;

```

The reader can be set automatically using **SecuBSP_DEVICE_ID_AUTO_DETECT**.

```

[Visual Basic]
objSecuBSP.OpenDevice(SecuBSP_DEVICE_ID_AUTO_DETECT)

```

```
[Delphi]
objSecuBSP.OpenDevice(SecuBSP_DEVICE_ID_AUTO_DETECT)
```

4.2.3. Closing the Device

The **CloseDevice** method should be used to close the reader. The same **DeviceID** used to call the **OpenDevice** method must be used again to call **CloseDevice**.

```
[Visual Basic]

DeviceID = SecuBSP_DEVICE_ID_FDP02_0
objSecuBSP.OpenDevice (DeviceID)
...
objSecuBSP.CloseDevice(DeviceID)
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Close device success ...
Else
    ' Close device failed ...
End If
```

```
[Delphi]
DeviceID := SecuBSP_DEVICE_ID_FDP02_0;
objSecuBSP.CloseDevice(DeviceID);
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Close device success ...
Else
    // Close device failed ...
```

The current reader must be closed before opening another reader.

4.2.4. Using Auto-On

Auto-On is a function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. To use this function, Auto-On should be enabled using **MonitorDevice()**. Once Auto-On is enabled, the application can receive a message from the device driver whenever an Auto-On event occurs in the device.

When calling **MonitorDevice()**, pass the handle of the window which will receive the Auto-On message. The Auto-On message is defined as **SecuAPI_WM_DEVICE_EVENT** (0x81000).

Note: Auto-On is not supported by FDP0x or FDU02- based readers.

- **Enabling Auto-On**

```
[Visual Basic]
objSecuBSP.MonitorDevice(True, Me.hwnd)

[Delphi]
m_ObjSecuBsp.MonitorDevice(true, Handle);
```

- **Disabling Auto-On**

```
[Visual Basic]
```

```
objSecuBSP.MonitorDevice(True, Me.hwnd)
```

```
[Delphi]
```

```
m_ObjSecuBsp.MonitorDevice(false, 0);
```

- **Handling AutoOn Event**

```
[Visual Basic]
```

```
Public Function MyWindowProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
```

```
    If (uMsg = SecuAPI_WM_DEVICE_EVENT) Then
        If (wParam = SecuAPI_DEVICE_FINGER_ON) Then
            frmMain.labStatus.Caption = "Device event: Finger On the Sensor"
        ElseIf (wParam = SecuAPI_DEVICE_FINGER_OFF) Then
            frmMain.labStatus.Caption = "Device event: Finger Off the Sensor"
        End If

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

```
End Function
```

```
[Delphi]
```

```
procedure OnAutoOnEvent(var Msg: TMessage); message SecuAPI_WM_DEVICE_EVENT;
```

```
procedure TForm1.OnAutoOnEvent(var Msg: TMessage);
begin
    if (Msg.WParam = SecuAPI_DEVICE_FINGER_ON) then
        begin
            txtStatus.Text := 'Device event: Finger On the Sensor';
        end
    else if (Msg.WParam = SecuAPI_DEVICE_FINGER_OFF) then
        begin
            txtStatus.Text := 'Device event: Finger off the Sensor';
        end
    end;
end;
```

4.3. Fingerprint Enrollment

The **Enroll** method is used to enroll fingerprints. All fingerprint data is used as the type of encoded text found in the **SecuBSP** module. Fingerprint data will be entered into the **FIRTextData** property upon successful enrollment. The **FIRTextData** has String type value.

```
[Visual Basic]
```

```
Dim szFIRTextData As String
```

```
...
```

```
objSecuBSP.Enroll(null)
```

```
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
```

```
    ' Enroll success ...
```

```
    szFIRTextData = objSecuBSP.FIRTextData
```

```
    ' Write FIR data to file or DB
```

```

Else
    ` Enroll failed ...
End If

[Delphi]
Var  szFIRTextData : WideString;
...
objSecuBSP.Enroll(null);
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
Else
    // Enroll failed ...

```

Fingerprint data will be stored when saving **FIRTextData** to a file or database.

4.4. Fingerprint Verification

The **Verify** method performs fingerprint verification by comparing the existing fingerprint data with newly input fingerprint data. The result is saved as a value in the **IsMatched** property: 1 for success and 0 for failed verification.

Values used in IsMatched

Define	Value
SecuBSP_TRUE	1
SecuBSP_FALSE	0

```

[Visual Basic]
Dim storedFIRTextData As String
...
` Read stored FIRText Data from File or DB.
...
objSecuBSP.Verify(storedFIRTextData)
If objSecuBSP.IsMatched = SecuBSP_TRUE then
    ` Verify success
Else
    ` Verify failed
End if

```

```

[Delphi]
Var  storedFIRTextData : String;
...
//Read FIRText Data from File or DB.
...
objSecuBSP.Verify(storedFIRTextData);
If objSecuBSP.IsMatched = SecuBSP_TRUE then
    // Verify success
Else
    // Verify failed

```


4.5. Client/Server Environment Programming

Unlike stand-alone environments, the fingerprint enrollment and matching processes occur in separate places within the Client/Server environment. Usually, fingerprints are enrolled in the client and subsequently matched in the server. The **Enroll** method registers fingerprints while the **Capture** method captures fingerprints for verification. The **VerifyMatch** method matches fingerprints in the server through the use of previously registered fingerprint data obtained from the client.

For more information on Client/Server programming, refer to [Chapter 3, SecuBSP Programming in C/C++](#).

4.5.1. Fingerprint Enrollment

Use the **Enroll** method for fingerprint enrollment in a client.

```
[Visual Basic]
Dim szFIRTextData As String
...
objSecuBSP.Enroll
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Enroll success ...
    szFIRTextData = objSecuBSP.FIRTextData
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If

[Delphi]
Var szFIRTextData : wideString;
...
objSecuBSP.Enroll;
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
Else
    //Enroll failed ...
```

4.5.2. Fingerprint Verification

Use the **Capture** method for registering one fingerprint in the client. While the **Enroll** method allows multiple fingerprints to be enrolled and transferred as a single FIR, the **Capture** method registers only one fingerprint.

```
[Visual Basic]
Dim szFIRTextData As String
...
objSecuBSP.Capture
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Capture success ...
    szFIRTextData = objSecuBSP.FIRTextData
    ' Write FIR data to file or DB
Else
    ' Capture failed ...
```

```

End If

[Delphi]
Var  szFIRTextData : WideString;
...
objSecuBSP.Capture;
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Capture success ...
    szFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
Else
    //Capture failed ...

```

The **VerifyMatch** method matches fingerprints stored on the server. See the **IsMatched** property to check results.

```

[Visual Basic]
Dim capturedFIRTextData1 As String
Dim storedFIRTextData2 As String
...
objSecuBSP.VerifyMatch(capturedFIRTextData1, storedFIRTextData2)
If objSecuBSP.IsMatched = SecuBSP_TRUE then
    ' Matching success
Else
    ' Matching failed
End if

[Delphi]
var
capturedFIRTextData1: String;
storedFIRTextData2 : String;
...
objSecuBSP.VerifyMatch(capturedFIRTextData1, storedFIRTextData2);
if objSecuBSP.IsMatched = SecuBSP_TRUE then
    // Matching success
else
    // Matching failed

```

4.5.3. Fingerprint Enrollment with Stored Fingerprint

When enrolling a fingerprint, a stored fingerprint may be used. With this feature, a fingerprint can be added to an existing stored template. To use the stored template, put the stored template data into the **StoredFIRTextData** property and then call the **Enroll** method. The enrollment wizard will show the position of the previously stored fingerprint and return **FIRTextData** that has been merged with the stored template data.

```

[Visual Basic]
Dim szFIRTextData As String
...

objSecuBSP.StoredFIRTextData = szFIRTextData
objSecuBSP.Enroll

```

```

If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Enroll success ...
    szFIRTextData = objSecuBSP.FIRTextData
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If

[Delphi]
Var szFIRTextData : wideString;
...
objSecuBSP.StoredFIRTextData := szFIRTextData;
objSecuBSP.Enroll;
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
Else
    //Enroll failed ...

```

4.6. Using Payload

Additional data, other than fingerprint data, that is included within the FIR is called Payload. Only encoded text type data can be used as Payload in the **SecuBSP** module.

For more information on Payload, refer to [Chapter 3. SecuBSP Programming in C/C++](#).

4.6.1. Inserting Payload into FIR

At the time of fingerprint enrollment, use the **Enroll** method to include **Payload** in the FIR. The **CreateTemplate** method can be used to insert **Payload** into an existing FIR. The **Enroll** method will use the fingerprint data and **Payload** to provide a parameter for later comparison.

```

[Visual Basic]
Dim szFIRTextData As String
Dim szPayload As String
...
szPayload = "Your Payload Data"
...
objSecuBSP.Enroll(szPayload)
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' Enroll success ...
    szFIRTextData = objSecuBSP.FIRTextData
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If

[Delphi]
Var
szFIRTextData : String;
szPayload : String;
...

```

```

szPayload := 'Your Payload Data';
...
objSecuBSP.Enroll(szPayload);
if objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
else
    // Enroll failed ...

```

Use the **CreateTemplate** method to insert **Payload** into an existing FIR. The **CreateTemplate** method can also add new fingerprint data into an existing FIR. Just as in the **Enroll** method, the new fingerprint data will be put into the **FIRTextData** property.

[Visual Basic]

```

Dim storedFIRTextData As String
Dim newFIRTextData As String
Dim szPayload As String
...
szPayload = "Your Payload Data"
...
objSecuBSP.CreateTemplate(storedFIRTextData, null, szPayload)
If objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    ' CreateTemplate success ...
    newFIRTextData = objSecuBSP.FIRTextData
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If

```

[Delphi]

```

Var
    storedFIRTextData : String;
    newFIRTextData : String;
    szPayload : String;
...
szPayload := 'Your Payload Data';
...
objSecuBSP.CreateTemplate(storedFIRTextData, null, szPayload);
if objSecuBSP.ErrorCode = SecuBSPERROR_NONE Then
    // CreateTemplate success ...
    newFIRTextData := objSecuBSP.FIRTextData;
    // Write FIR data to file or DB
else
    // Enroll failed ...

```

4.6.2. Extracting Payload from FIR

Payload in a FIR (registered data) will be extracted only if matched using the **Verify** method or if the **VerifyMatch** method returns true.

Check the **IsPayload** property after matching to verify whether **Payload** exists. If **IsPayload** is true, then

Payload will be shown in the **PayloadData** property.

[Visual Basic]

```
Dim storedFIRTextData As String
Dim szPayload As String
...
' Read FIRText Data from File or DB.
...
objSecuBSP.Verify(storedFIRTextData)
If objSecuBSP.IsMatched = SecuBSP_TRUE Then
    ' Verify success
    If objSecuBSP.IsPayload = SecuBSP_TRUE Then
        ' Exist payload
        szPayload = objSecuBSP.PayloadData
    End If
Else
    ' Verify failed
End if
```

[Delphi]

```
Var
    storedFIRTextData : String;
    szPayload : String;
...
// read FIRText data from File or DB.
...
objSecuBSP.Verify(storedFIRTextData);
if objSecuBSP.IsMatched = SecuBSP_TRUE Then
    // Verify success
    begin
        if objSecuBSP.IsPayload = SecuBSP_TRUE Then
            // Exist payload
            szPayload := objSecuBSP.PayloadData;
        end
    end
else
    // verification failed
```

Payloads are extracted by the **VerifyMatch** method in the same way as the **Verify** method. When calling **VerifyMatch**, use the data to be compared as a first parameter, and use stored data (enrolled template) as a second parameter. Payload data can only be extracted from the FIR in the second parameter (**enrolledFIRTextData**). Even if FIR data in the first parameter includes payload, it will not be retrieved.

[Visual Basic]

```
...
objSecuBSP.VerifyMatch(capturedFIRTextData, enrolledFIRTextData)
if objSecuBSP.IsMatched = SecuBSP_TRUE then
    'Verify success
    if objSecuBSP.IsPayload = SecuBSP_TRUE then
        'Get payload
        szPayload = objSecuBSP.PayloadData
    end if
end if
```

[Delphi]

```
objSecuBSP.VerifyMatch(capturedFIRTextData, storedFIRTextData);
```

```

if objSecuBSP.IsMatched = SecuBSP_TRUE Then
    // Verify success
    begin
        if objSecuBSP.IsPayload = SecuBSP_TRUE Then
            // Exist payload
            szPayload := objSecuBSP.PayloadData;
        end
    else
        // verification failed
    end

```

4.7. Loading the SecuBSP Resource file

The **SecuBSP COM** module offers resource files for customizing the basic user interface (UI) in English. Use the **SetSkinResource** method to load UI resources in languages other than English. Resource files must have an absolute path. Separate documents are available for making customized UIs.

```

[Visual Basic]
Dim szSkinPath
...
CommonDialog.ShowOpen

If CommonDialog.CancelError = False then
    szSkinPath = CommonDialog.FileName
    objSecuBSP.SetSkinResource(szSkinPath)
End if

[Delphi]
szSkinFileName : String; ...
if OpenFileDialog1.Execute then
    begin
        szSkinFileName := OpenFileDialog1.FileName;
        // Set skin resource
        ret := objSecuBSP.SetSkinResource(szSkinFileName) ;
    end
end

```

4.8. Changing the Window Style

In version 2.50 and higher, the **SecuBSP COM** module allows you to change the window style for the enrollment or verification wizards. Before calling **Enroll()** or **Verify()**, specify **WindowStyle**.

Enrollment

The Welcome page shown during the enrollment process can be removed.

```

[Visual Basic]
' Set Window Style
objSecuBSP.WindowStyle = WS_ENROLL_NO_WELCOME PAGE
' Enroll
Call objSecuBSP.Enroll(szUserID)

[Delphi]
// Set Window Style
objSecuBSP.WindowStyle = WS_ENROLL_NO_WELCOME PAGE;
// Enroll

```

```
objSecuBSP.Enroll(szUserID);
```

Verification

The verification window shown during the verification process can be hidden. This may be useful for applications needing very high levels of privacy. Using this option, since the verification window is not displayed, users' fingerprints are not revealed on the screen while fingerprints are captured.

[Visual Basic]

```
` Set Window Style
objSecuBSP.WindowStyle = WS_CAPTURE_INVISIBLE
' Verify
Call objSecuBSP.Verify(szFIRTextData)
```

[Delphi]

```
// Set Window Style
objSecuBSP.WindowStyle = WS_CAPTURE_INVISIBLE;
// Verify
objSecuBSP.Verify(szFIRTextData);
```

4.9. ANSI378 Format Fingerprint Data

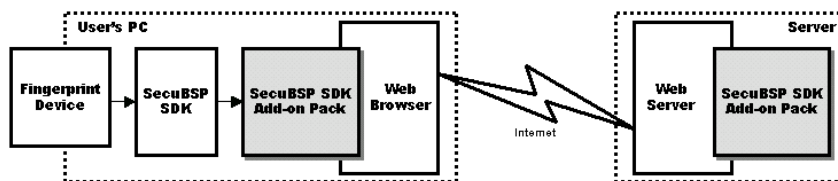
After enrolling or capturing a fingerprint, use the **FIRFormat** property to retrieve fingerprint data in the ANSI378 Format. For more information about the ANSI378 format, refer to [Section 3.11. ANSI378 Format Fingerprint Data](#).

```
objSecuBSP.FIRFormat = SecuAPI_FIR_FORMAT_ANSI378
```

Chapter 5. SecuBSP COM Programming in ASP

The **SecuBSP COM** module runs on web environments without any additional modification due to the fact that it is built on the Microsoft COM architecture. It can also be used simultaneously on servers and clients because it includes the server component and client Active X functions.

Module Structure in Web Environments



5.1. Registration

User fingerprints must be registered through a web browser for user enrollment. The fingerprint data will be saved either in a file or database on the server.

5.1.1. Code for Setting Object

The SecuBSP COM module should be set as an object for each HTML page to be used on a web browser.

```
<OBJECT classid="CLSID:D84A0B2D-912D-4570-B99A-8FCAA6F8CA01"
        codebase="objSecuBSP.cab#version=1,0,0,1"
        height=0 width=0
        id="objSecuBSP"
        name="objSecuBSP">

</OBJECT>
```

The name shown here will be used as the object's name in JavaScript.

5.1.2. Form for Transferring Fingerprint Information

Fingerprint data registered in JavaScript is transferred to the server in this form.

```
<form action='regist.asp' name='MainForm' method='post' OnSubmit='return regist();'>
<input type=hidden name='FIRTextData'>
User ID : <input type=text name=UserID size=20><p>
<input type=submit value=' Click here to register your fingerprint '>
</form>
```

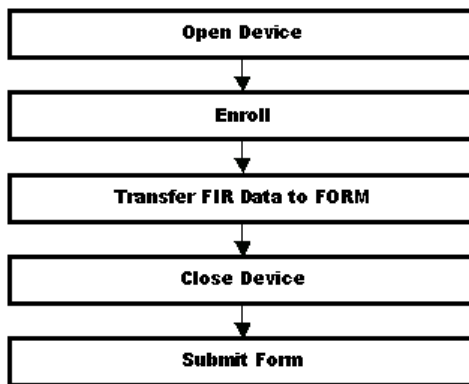
Upon selecting the form, JavaScript calls the **regist()** function and performs the registration. Fingerprint data

registered in this way will be transferred to the texts of **FIRTextData**.

5.1.3. JavaScript Code for Fingerprint Registration

JavaScript code will be used to communicate between the web browser and the **SecuBSP COM** module. The **Enroll** method also can be used.

Process flow in the Regist function



The example above shows how functions can be composed. Note that JavaScript is case sensitive.

```
<script lang='javascript'>
```

```
function regist()
{
    var err, payload

    // Check ID is not NULL
    if ( document.MainForm.UserID.value == '' )
    {
        alert('Please enter user id !');
        return(false);
    }

    try // Exception handling
    {
        // Open device. [AUTO_DETECT]
        // You must open device before enroll.
        DEVICE_FDP02      = 1;
        DEVICE_FDU02      = 2;
        DEVICE_AUTO_DETECT = 255;

        document.objSecuBSP.OpenDevice(DEVICE_AUTO_DETECT);

        err = document.objSecuBSP.ErrorCode; // Get error code

        if ( err != 0 )                // Device open failed
        {
            alert('Device open failed !');
            return(false);
        }
    }
}
```

```

    }

    // Enroll user's fingerprint.
    document.objSecuBSP.Enroll(payload);
    err = document.objSecuBSP.ErrorCode; // Get error code

    if ( err != 0 )                // Enroll failed
    {
        alert('Registration failed ! Error Number :
              [' + err + ']');
        document.objSecuBSP.CloseDevice(255);
        return(false);
    }
    else // Enroll success
    {
        // Get text encoded FIR data from SecuBSP module.
        document.MainForm.FIRTextData.value =
            document.objSecuBSP.FIRTextData;
        alert('Registration success !');
    }

    // Close device. [AUTO_DETECT]
    document.objSecuBSP.CloseDevice(DEVICE_AUTO_DETECT);

}
catch(e)
{
    alert(e.message);
    return(false);
}

// Submit main form
document.MainForm.submit();
return(true);
}

</script>

```

5.1.4. Storing Fingerprint Information

Perform verification by bringing out fingerprint data using ASP code. Save it in either a file or database on the server.

```

<%
UserID = Request.Form("UserID")
FIRTextData = Request.Form("FIRTextData")

` Write UserID and FIRTextData to File or DB.
%>

```

5.2. Verification

Fingerprints captured for verification will be transferred and compared to the fingerprint data stored on the server.

5.2.1. Code for Setting Object

The **SecuBSP COM** module should be set as an object for each HTML page to be used on the web browser.

```
<OBJECT classid="CLSID:D84A0B2D-912D-4570-B99A-8FCAA6F8CA01"
        codebase="objSecuBSP.cab#version=1,0,0,1"
        height=0 width=0
        id="objSecuBSP"
        name="objSecuBSP">

</OBJECT>
```

The name shown here will be used for the object in JavaScript.

5.2.2. Form for Transferring Fingerprint Information

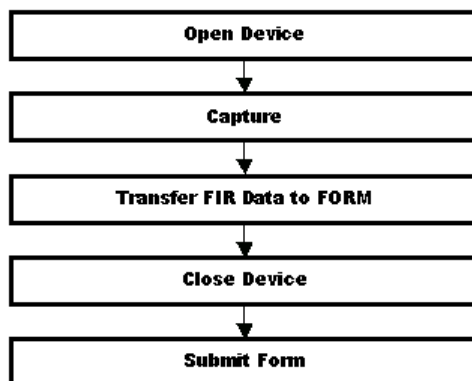
Fingerprint data captured in JavaScript will be transferred to the server in this form. The JavaScript function for capturing fingerprint is called when the form is submitted.

```
<form action='verify.asp' name='MainForm' method='post' OnSubmit='return capture();'>
<input type=hidden name='FIRTextData'>
User ID : <input type=text name=UserID size=20>
<p>
<input type=submit value=' Click here to verify with your fingerprint '>
</form>
```

Upon selecting the button set in this form, the fingerprint will be captured as if calling the **capture()** function in JavaScript and then transferred as **FIRTextData** text.

5.2.3. JavaScript Code for Capturing Fingerprints

Process flow in the capture function



The flow chart above shows how functions can be composed. Note that text is case sensitive in JavaScript.

```
<script lang='javascript'>

function capture()
{
var err

// Check ID is not NULL
if ( document.MainForm.UserID.value == '' )
{
alert('Please enter user id !');
return(false);
}

try // Exception handling
{
    // Open device. [AUTO_DETECT]
    // You must open device before capture.
    DEVICE_FDP02    = 1;
    DEVICE_FDU02    = 2;
    DEVICE_AUTO_DETECT    = 255;

    document.objSecuBSP.OpenDevice(DEVICE_AUTO_DETECT);
    err = document.objSecuBSP.ErrorCode; // Get error code

    if ( err != 0 )                // Device open failed
    {
        alert('Device open failed !');
        return(false);
    }

    // Enroll user's fingerprint.
    document.objSecuBSP.Capture();
    err = document.objSecuBSP.ErrorCode; // Get error code

    if ( err != 0 )                // Enroll failed
    {
        alert('Capture failed ! Error Number : [' + err + ']');
        document.objSecuBSP.CloseDevice(DEVICE_AUTO_DETECT);
        return(false);
    }
    else // Capture success
    {
        // Get text encoded FIR data from SecuBSP module.
        document.MainForm.FIRTextData.value=
            document.objSecuBSP.FIRTextData;
        alert('Capture success !');
    }

    // Close device. [AUTO_DETECT]
    document.objSecuBSP.CloseDevice(DEVICE_AUTO_DETECT);
} // end try

catch(e)
{
    alert(e.message);
}
```

```

        return(false);
    }

    // Submit main form
    document.MainForm.submit();
    return(true);
}

</script>

```

5.2.4. Matching against an Existing Fingerprint

Use the **verifyMatch** method to read existing fingerprint data and compare it to the input (query) fingerprint data. The result can be found in the **IsMatched** property.

```

<%
' Read FIR data from file or DB.

Set objSecuBSP = Server.CreateObject("SecuBSPCOM.APIInterface")

' Verify Match
' FIRTextData is Caputured FIR
' fFIRTextData is FIR from file
err = objSecuBSP.VerifyMatch(FIRTextData, fFIRTextData)

if ( objSecuBSP.IsMatched = 0 ) then ' Matching failed. [0:FALSE,Others:TRUE]
    ' Matching failed ! Verification failed !
else ' Matching success
    ' Verification success !!!
end if

' Release SecuBSP object
Set objSecuBSP = nothing
%>

```

Chapter 6. SecuBSP .NET Programming

The SecuBSP .NET library (SecuBSPMx.NET.DLL) is designed for .NET application developers to easily use SecuBSP functionalities. All SDK functions are implemented in the SecuBSP class in SecuBSPMx.NET.DLL. This chapter explains how to use the SecuBSP class in your .NET application. Since the SecuBSP .NET library calls SecuBSPMx.DLL, SecuBSPMx.DLL should be in the execution path. To use SecuBSPMx.NET.DLL, .NET Framework 2.0 SP1 is required.

Requirements:

- .NET framework 2.0 SP1
- SecuBSPMx.DLL, SecuBSPMx.NET.DLL

6.1. SecuBSPMx.NET.DLL and Namespace

6.1.1. Include SecuBSPMx.NET.DLL

To use the SecuBSP .NET library (SecuBSP class), include the SecuBSPMx.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 SecuBSPMx.NET.DLL). The SecuBSPMx.NET.DLL file is located in the Bin directory of the SDK installation directory.

6.1.2. Namespace

The SecuBSP class is in the SecuGen.SecuBSPSDK.Windows namespace. Specify the SecuGen.SecuBSPSDK.Windows namespace in the source file where the SecuBSP class is used.

```
[C#]
using SecuGen.SecuBSPSDK.Windows;

[VB.NET]
Imports SecuGen.SecuBSPSDK.Windows
```

6.2. Create SecuBSP Class

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

```
[C#]
private SecuBSP      m_SecuBSP; // member variable
...
m_SecuBSP = new SecuBSP();

[VB.NET]
Dim m_SecuBSP As SecuBSP 'member variable
...
m_SecuBSP = New SecuBSP()
```

6.3. Device Functions

6.3.1. Opening the device

The **OpenDevice()** method is used to initialize the reader. Before calling this method, the device ID should be set using the **DeviceID** property. If **DeviceID** is not set, **DeviceID.AUTO** will be set as the default. If **DeviceID** is **DeviceID.AUTO**, then SecuBSP will enumerate the device and then use the first enumerated device as the current device.

DeviceID has a 2 byte-length. The lower byte represents the device name, and the higher byte represents its instance number. **FDP01**- and **FDP02**-based (parallel) fingerprint readers have **0x01** as the device name. **FDU02**-based (USB) fingerprint readers have **0x02** as the device name. **FDU03**- and **SDU03**-based (USB) fingerprint readers have **0x03** as the device name. **FDU04**-based (USB) fingerprint readers have **0x04** as the device name. The instance number starts from 0. If there is only one device for each device type in the system, the instance number will be '0.' If there are two devices that have the same device type, the instance number will be '0' and '1' for each device.

For example, the DeviceID for the first instance of an FDP02-based reader will be 0x0001. The DeviceID for the second instance of an FDP02-based reader will be 0x0101. In the same way, the DeviceID for the first instance of an FDU02-based reader will be 0x0002.

$$\text{DeviceID} = \text{Instance Number} + \text{Device Name}$$

Device Name

Device Name	Value	Notes
DeviceName.FDP02	0x01	FDP01 & FDP02 parallel readers
DeviceName.FDU02	0x02	FDU02 USB readers
DeviceName.FDU03	0x03	FDU03 & SDU03 USB readers
DeviceName.FDU04	0x04	FDU04 USB readers

Predefined Device Ids

Device ID	Value	Notes
DeviceID.UNKNOWN	0(0x0000)	No devices
DeviceID.PARALLEL_0	1(0x0001)	The first instance of FDP01 or FDP02
DeviceID.USB_0	2(0x0002)	The first instance of FDU02
DeviceID.USB_1	3(0x0003)	The first instance of FDU03 or SDU03
DeviceID.AUTO	255(0x00FF)	Detect device automatically

If you are unsure about which readers have been installed, use the **Enumerate** method to get information about the connected readers.

The **OpenDevice()** method should be called before the using device related functions, such as enrollment, verification, and capture.

[C#]

```
m_SecuBSP.DeviceID = DeviceName.FDU02;
err = m_SecuBSP.OpenDevice();
```

[VB.NET]

```
m_SecuBSP.DeviceID = DeviceName.FDU02
err = m_SecuBSP.OpenDevice()
```

6.3.2. Enumerate devices

If it is not known which readers are currently being used, before opening the device, use the **EnumerateDevice()** method to get information about the number and type(s) of reader(s) linked to the PC. Once this is called, the number of readers linked to the PC will appear in the **DeviceNum** property. You can get the **DeviceID** for each reader using the **GetDeviceID()** method.

The following is an example of how to use the **EnumerateDevice()** method.

[C#]

```
err = m_SecuBSP.EnumerateDevice();
if (err == BSPError.ERROR_NONE)
{
    for (int i = 0; i < m_SecuBSP.DeviceNum; i++)
    {
        Int16 device_id = m_SecuBSP.GetDeviceID(i);
        string device_id_info;
        device_id_info = "0x" + MakeHexadecimal(device_id, 4) + " ("
            + m_SecuBSP.GetDeviceName(device_id) + ","
            + m_SecuBSP.GetDeviceInstanceNum(device_id) + ")";

        DeviceIDCombo.Items.Add(device_id_info);
    }
}
```

[VB.NET]

```
Dim device_id As Int16
Dim device_id_info As String
Dim i As Integer
Dim device_name As String
Dim device_instance As String

err = m_SecuBSP.EnumerateDevice()
If (err = BSPError.ERROR_NONE) Then
    For i = 0 To m_SecuBSP.DeviceNum

        device_id = m_SecuBSP.GetDeviceID(i)
        device_name = Convert.ToString(m_SecuBSP.GetDeviceName(device_id))
        device_instance = Convert.ToString(m_SecuBSP.GetDeviceInstanceNum(device_id))
        device_id_info = "0x" + MakeHexadecimal(device_id, 4) + " (" _
            + Convert.ToString(m_SecuBSP.GetDeviceName(device_id)) + "," _
            + Convert.ToString(m_SecuBSP.GetDeviceInstanceNum(device_id)) + ")"

        DeviceIDCombo.Items.Add(device_id_info)
    Next
End If
```


6.3.3. Closing the Device

The **CloseDevice()** method should be used to close the reader. The current reader must be closed before opening another reader.

```
[C#]
err = m_SecuBSP.CloseDevice();

[VB.NET]
err = m_SecuBSP.CloseDevice()
```

6.3.4. Using Auto-On

Auto-On is a function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. To use this function, Auto-On should be enabled using **MonitorDevice()**. Once Auto-On is enabled, the application can receive a message from the device driver whenever an Auto-On event occurs in the device.

When calling **MonitorDevice()**, pass the handle of the window which will receive the Auto-On message. The Auto-On message is defined as **SecuAPI_WM_DEVICE_EVENT** (0x81000).

Note: Auto-On is not supported by FDP0x or FDU02-based readers.

- **Enabling Auto-On**

```
[C#]
m_SecuBSP.MonitorDevice(true, (int)this.Handle);

[VB.NET]
m_SecuBSP.MonitorDevice(True, Me.Handle.ToInt32())
```

- **Disabling Auto-On**

```
[C#]
m_SecuBSP.MonitorDevice(false, 0);

[VB.NET]
m_SecuBSP.MonitorDevice(False, 0)
```

- **Handling an Auto-On Event**

```
[C#]
protected override void WndProc(ref Message message)
{
    if (message.Msg == (int)DriverMessage.WM_DEVICE_EVENT)
    {
        if (message.WParam.ToInt32() == (Int32)DeviceEvent.FINGER_ON)
            StatusBar.Text = "Device Message: Finger On";
        else if (message.WParam.ToInt32() == (Int32)DeviceEvent.FINGER_OFF)
            StatusBar.Text = "Device Message: Finger Off";
    }
    base.WndProc(ref message);
}
```

```

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

```

6.4. Fingerprint Enrollment

The **Enroll()** method is used to enroll fingerprints. All fingerprint data is extracted as a type of encoded text. Fingerprint data will be entered into the **FIRTextData** property upon successful enrollment. The **FIRTextData** has a string type value.

```

[C#]
string m_EnrollFIRText;
err = m_SecuBSP.Enroll("SecuGen");
if (err == BSPError.ERROR_NONE)
{
    m_EnrollFIRText = m_SecuBSP.FIRTextData;
}

[VB.NET]
err = m_SecuBSP.Enroll(textUserID.Text)
If (err = BSPError.ERROR_NONE) Then
    m_EnrollFIRText = m_SecuBSP.FIRTextData
End If

```

6.5. Fingerprint Verification

The **Verify()** method performs fingerprint verification by comparing the existing fingerprint data with newly input fingerprint data. The result is saved as a value in the **IsMatched** property: **true** for verification success and **false** for verification failure.

```

[C#]
if (m_EnrollFIRText != "")
{
    err = m_SecuBSP.Verify(m_EnrollFIRText);

    if (err == BSPError.ERROR_NONE)
    {
        if (m_SecuBSP.IsMatched)
            // Verification Success
        else
            // Verification Fail
    }
}

[VB.NET]

```

```

If (m_EnrollFIRText <> "") Then
    err = m_SecuBSP.Verify(m_EnrollFIRText)
    If (err = BSPError.ERROR_NONE) Then
        If (m_SecuBSP.IsMatched) Then
            'Verification Success
        Else
            'Verification Fail
        End If
    End If
End If
End If

```

6.6. Client/Server Environment Programming

Unlike stand-alone environments, the fingerprint enrollment and matching processes occur in separate locations within the Client/Server environment. Usually, fingerprints are enrolled in the client and subsequently matched in the server. The **Enroll()** method registers fingerprints while the **Capture()** method captures fingerprints for verification. The **VerifyMatch()** method matches fingerprints in the server through the use of previously registered fingerprints obtained through the client.

For more information on Client/Server programming, refer to [Chapter 3, SecuBSP Programming in C/C++](#).

6.6.1. Fingerprint Enrollment

Use the **Enroll()** method for acquiring enrollment data in the client. Use the **Capture()** method for acquiring data for verification in the client.

```

[C#]
string m_EnrollFIRText;
...
err = m_SecuBSP.Enroll(textUserID.Text);
if (err == BSPError.ERROR_NONE)
    m_EnrollFIRText = m_SecuBSP.FIRTextData;

[VB.NET]
Dim m_EnrollFIRText As string
...
err = m_SecuBSP.Enroll(textUserID.Text)
if (err = BSPError.ERROR_NONE) then
    m_EnrollFIRText = m_SecuBSP.FIRTextData

```

6.6.2. Capturing a Fingerprint

Use the **Capture()** method for acquiring data for verification in the client. You should specify the capture purpose as **FIRPurpose.VERIFY**. While the **Enroll()** method allows multiple fingerprints to be enrolled and transferred as a single FIR, the **Capture()** method can capture only one fingerprint if **FIRPurpose** is **VERIFY**.

```

[C#]
string m_CaptureFIRText;
...
err = m_SecuBSP.Capture(FIRPurpose.VERIFY);
if (err == BSPError.ERROR_NONE)

```

```

        m_CaptureFIRText = m_SecuBSP.FIRTextData;

[VB.NET]
Dim m_CaptureFIRText As string
...
err = m_SecuBSP.Capture(FIRPurpose.VERIFY)
if (err = BSPError.ERROR_NONE) then
    CaptureFIRText = m_SecuBSP.FIRTextData

```

6.6.3. Verifying a Fingerprint

The **VerifyMatch()** method performs matching using two FIR data fingerprints stored on the server. Use **IsMatched** property to check the results.

```

[C#]
err = m_SecuBSP.VerifyMatch(m_CaptureFIRText, m_EnrollFIRText);
if (err == BSPError.ERROR_NONE)
{
    if (m_SecuBSP.IsMatched)
        // Matched
    else
        // Not Matched
}

[VB.NET]
err = m_SecuBSP.VerifyMatch(m_CaptureFIRText, m_EnrollFIRText)
If (err = BSPError.ERROR_NONE) then
If (m_SecuBSP.IsMatched) then
    ` Matched
Else
    ` Not Matched
End If
End If

```

6.7. Using Payload

Additional data, other than fingerprint data, that is included within the FIR is called Payload. Only text data can be used as Payload. For more information on payload, refer to [Chapter 3, SecuBSP Programming in C/C++](#).

6.7.1. Inserting Payload into FIR

At the time of fingerprint enrollment - **Enroll()** method, you can include payload in the FIR. Pass payload as a parameter in **Enroll()**. After **Enroll()**, the payload will be inserted in the **FIRTextData** property.

```

[C#]
err = m_SecuBSP.Enroll("SecuGen"); // insering payload "SecuGen"
if (err == BSPError.ERROR_NONE)
{
    m_EnrollFIRText = m_SecuBSP.FIRTextData;
}

```

```

[VB.NET]
err = m_SecuBSP.Enroll("SecuGen") ` inserting payload "SecuGen"
if (err = BSPError.ERROR_NONE) Then
    m_EnrollFIRText = m_SecuBSP.FIRTextData
End If

```

Use the **CreateTemplate()** method to insert payload into existing fingerprint data. The **CreateTemplate()** method can also add new fingerprint data into an existing FIR. Just as in the **Enroll()** method, the new fingerprint data will be put into the **FIRTextData** property.

```

[C#]
err = m_SecuBSP.CreateTemplate(m_EnrollFIRText, m_CaptureFIRText, "SecuGen");
string newfirtext = m_SecuBSP.FIRTextData;

[VB.NET]
Dim newfirtext As String

err = m_SecuBSP.CreateTemplate(m_EnrollFIRText, m_CaptureFIRText, "SecuGen")
newfirtext = m_SecuBSP.FIRTextData

```

6.7.2. Extracting Payload from FIR

The payload in a FIR (enrollment data) will only be extracted if verification is successful. After matching, check the **IsPayload** property to know if payload exists. If **IsPayload** is true, the payload will be shown in the **PayloadData** property.

```

[C#]
err = m_SecuBSP.Verify(m_EnrollFIRText);

if (m_SecuBSP.IsMatched)
{
    if (m_SecuBSP.IsPayload)
        MessageBox.Show(m_SecuBSP.PayloadData, "Payload data found",
                        MessageBoxButtons.OK);
}

[VB.NET]
err = m_SecuBSP.Verify(m_EnrollFIRText)

If (m_SecuBSP.IsMatched) Then
If (m_SecuBSP.IsPayload) Then
    MessageBox.Show(m_SecuBSP.PayloadData, "Payload data found",
                    MessageBoxButtons.OK);
End If
End If

```

Payloads are extracted by the **VerifyMatch** method in the same way as the **Verify** method. When calling **VerifyMatch**, use the data to be compared as a first parameter, and use stored data (enrolled template) as a second parameter. Payload data can only be extracted from the FIR in the second parameter. Even if FIR data in the first parameter includes payload, it will not be retrieved.

6.8. Loading the SecuBSP Resource file

The SecuBSP SDK offers resource files for customizing the basic user interface (UI) in English. Using the SecuGen UI customization tool, you can make your own resource file. Use the **SetSkinResource()** method to load customized UI resources. Resource files must have an absolute path. Separate documents are available for making customized UIs.

```
[C#]
m_SecuBSP.SetSkinResource("c:\\windows\\system32\\myskin.dll");

[VB.NET]
m_SecuBSP.SetSkinResource("c:\\windows\\system32\\myskin.dll");
```

6.9. Changing the Window Style

The window style shown during enrollment and verification/capture may be changed using the **EnrollWindowOption** and **CaptureWindowOption** fields, respectively. To effect a change in the wizard, this field should be set before calling **Enroll()**, **Verify()** or **Capture()**.

6.9.1. EnrollWindowOption

EnrollWindowOption affects the enrollment wizard by hiding or showing the Welcome Screen during enrollment.

```
[C#]
m_SecuBSP.EnrollWindowOption.WelcomePage = false;
err = m_SecuBSP.Enroll(textUserID.Text);

[VB.NET]
m_SecuBSP.EnrollWindowOption.WelcomePage = false
err = m_SecuBSP.Enroll(textUserID.Text)
```

6.9.2. CaptureWindowOption

CaptureWindowOption affects the verification/capture wizard in the following three ways:

WindowStyle

With this property, the capture window shown during verification can be hidden. This may be useful for applications requiring very high levels of privacy. With the verification window hidden, the user's fingerprint will not be displayed on the screen while the fingerprint is captured.

ShowFPImage

With this property, the fingerprint image shown during verification/capture can be hidden. This may be useful for applications requiring very high levels of privacy. With the fingerprint image hidden, the user's fingerprint will not be displayed on the screen while the fingerprint is captured.

FingerWindow

With this option, the application's window can be used for displaying the fingerprint. To do this, **WindowStyle** should be **WindowStyle.INVISIBLE**.

```
[C#]
```

```
// hide the capture window
m_SecuBSP.CaptureWindowOption.WindowStyle = (int)WindowStyle.INVISIBLE;
err = m_SecuBSP.Capture(FIRPurpose.VERIFY);

// hide the fingerprint image
m_SecuBSP.CaptureWindowOption.ShowFPImage = false;
err = m_SecuBSP.Capture(FIRPurpose.VERIFY);

// Use my window for showing fingerprint image
m_SecuBSP.CaptureWindowOption.WindowStyle = (int)WindowStyle.INVISIBLE;
m_SecuBSP.CaptureWindowOption.FingerWindow = (int)fpWindow.Handle;
err = m_SecuBSP.Capture(FIRPurpose.VERIFY);

[VB.NET]
// hide the capture window
m_SecuBSP.CaptureWindowOption.WindowStyle = WindowStyle.INVISIBLE
err = m_SecuBSP.Capture(FIRPurpose.VERIFY)

// hide the fingerprint image
m_SecuBSP.CaptureWindowOption.ShowFPImage = false
err = m_SecuBSP.Capture(FIRPurpose.VERIFY)

// Use my window for showing fingerprint image
m_SecuBSP.CaptureWindowOption.WindowStyle = WindowStyle.INVISIBLE
m_SecuBSP.CaptureWindowOption.FingerWindow = fpWindow.Handle.ToInt32()
err = m_SecuBSP.Capture(FIRPurpose.VERIFY)
```

6.10. Data Conversion Functions

6.10.1. Importing FDx Data

To import 400 byte fingerprint minutiae formatted data (from FDx SDK) and convert into the SecuBSP FIR format, use **ImportFDxData()**. The FDx data import sample program can be found in SecuBSP.NET\Samples\bspdemo_advanced_cs\FDxToSecuBSP in the SecuBSP SDK directory.

```
[C#]
BSPError err = m_SecuBSP.ImportFDxData(m_TestMin,
                                         FDxMinType.MIN_TYPE_SG400, FIRPurpose.VERIFY);

if (err == 0)
{
    m_ConvertedFIRText = m_SecuBSP.FIRTextData;
}
else
    StatusBar.Text = "Conversion Failed";
```

6.10.2. Exporting FDx Data

To export SecuBSP FIR format data and convert into 400 byte fingerprint minutiae formatted data (to use with FDx SDK), use **SecuAPI_ExportFDx ()**. The exported FDx Data is contained in the **ExportMinutiaeDataStruct** class. For more information about **ExportMinutiaeDataStruct**, refer

to [Appendix B. SecuBSP® .NET Reference](#). The FDx data export sample program can be found in SecuBSP.NET\Samples\bspdemo_advanced_cs\SecuBSPtoFDx in the SecuBSP SDK directory.

```
[C#]
err = m_SecuBSP.ExportFDxData(m_TestFIR, FDxMinType.MIN_TYPE_SG400);
if (err == BSPError.ERROR_NONE)
{
    m_FIRMinutiaeData = m_SecuBSP.FIRMinutiaeData;

    NumFingsLabel.Text = "Number of Fingers: " +
    Convert.ToString(m_FIRMinutiaeData.NumOfFingers);
    NumSamplesLabel.Text = "Samples per Finger: " +
    Convert.ToString(m_FIRMinutiaeData.SamplesPerFinger);
    for (int i = 0; i < m_FIRMinutiaeData.NumOfFingers; i++)
    {
        Byte finger_pos = (Byte)m_FIRMinutiaeData.MinutiaeData[i].FingerPos;
        m_RadioButton[finger_pos].Enabled = true;
    }
}
```

6.11. Audit Data

During the enrollment or verification process, audit data may be obtained for later use. Audit data contains the fingerprint image for each enrolled fingerprint. To get audit data during enrollment or verification, set the **EnableAuditData** property to True. To get audit data after enrollment or verification, use **AuditFIRTextData**. To get image data from audit data, use the **ExportAuditData()** method, which will return finger information from the FIR and image data for each finger in the **ExportImageDataStruct** field. For more information about **ExportImageDataStruct**, refer to [Appendix B. SecuBSP® .NET Reference](#). The audit data export sample program can be found in SecuBSP.NET\Samples\bspdemo_advanced_cs\ExportAuditData in the SecuBSP SDK directory.

```
[C#]
err = m_SecuBSP.ExportAuditData(m_AuditFIR);
if (err == BSPError.ERROR_NONE)
{
    m_FIRImageData = m_SecuBSP.FIRImageData;

    NumFingsLabel.Text = "Number of Fingers: " +
    Convert.ToString(m_FIRImageData.NumOfFingers);
    NumSamplesLabel.Text = "Samples per Finger: " +
    Convert.ToString(m_FIRImageData.SamplesPerFinger);
    ImageWidthLabel.Text = "Image Width: " + Convert.ToString(m_FIRImageData.ImageWidth);
    ImageHeightLabel.Text = "Image Height: " +
    Convert.ToString(m_FIRImageData.ImageHeight);

    for (int i = 0; i < m_FIRImageData.NumOfFingers; i++)
    {
        Byte finger_pos = (Byte)m_FIRImageData.ImageData[i].FingerPos;
        m_RadioButton[finger_pos].Enabled = true;
    }
}
```


6.12. ANSI378 Format Fingerprint Data

To get FIR data in ANSI378 format, use the SetFIRFormat() method. SetFIRFormat() can take FIRFormat.ANSI378 or FIRFormat.STANDARDPRO. If SetFIRFormat() is not used, then STANDARDPRO will be the default FIR format. To change the FIR format, SetFIRFormat() should be called before calling the Enroll() or Capture() function.

```
FIRFormat format;  
  
if (ckboxAnsi378.Checked)  
    format = FIRFormat.ANSI378;  
else  
    format = FIRFormat.STANDARDPRO;  
  
m_SecuBSP.SetFIRFormat(format);
```

Appendix A. SecuBSP COM Reference

A.1. Property

BSTR Version

Gets SecuBSPMx.DLL version

- **Remarks**
Read only

LONG ErrorCode

Records the error code used in the last method. Refer to the *SecuAPI Reference Manual* for error codes.

- **Remarks**
Read only

LONG CaptureTimeout

Sets the timeout value upon verification (default is 10,000 ms)

LONG EnrollImageQuality

Controls the quality of the image captured in enrollment (ranges from 0 to 100, with default of 50)

LONG VerifyImageQuality

Controls the quality of the image captured in verification (ranges from 0 to 100, with default of 30)

LONG SecurityLevel

Indicates the security level set for fingerprint recognition (ranges from 1 (lowest) to 9 (highest), with default of 5 (normal))

- 1 = LOWEST
- 2 = LOWER
- 3 = LOW
- 4 = BELOW NORMAL
- 5 = NORMAL
- 6 = ABOVE NORMAL
- 7 = HIGH
- 8 = HIGHER
- 9 = HIGHEST

LONG DeviceNum

Indicates the number of readers linked to the PC. This must be used after calling the **EnumerateDevice()** function.

- **Remarks**
Read only

LONG DeviceID (LONG IDDeviceID)

Provides information regarding the types of readers linked to the PC. Upon designating a specific device number (beginning with 0), it provides the Device ID for that number. This must be used after calling the **EnumerateDevice()** function.

Device ID	Value
SecuBSP_DEVICE_ID_FDP02_0	0x0001
SecuBSP_DEVICE_ID_FDU02_0	0x0002
SecuBSP_DEVICE_ID_FDU03_0	0x0003
SecuBSP_DEVICE_ID_FDU04_0	0x0004
SecuBSP_DEVICE_ID_AUTO_DETECT	0x00FF

- Remarks**

Read only

LONG FIRFormat

Sets the format for FIR Data. SecuBSP SDK *Pro* has two formats:

SecuBSP_FIR_FORMAT_STANDARDPRO: Default format; FIR is encrypted

SecuBSP_FIR_FORMAT_ANSI378: ANSI378 format; FIR is not encrypted

SecuBSP_FIR_FORMAT_STANDARDPRO 3

SecuBSP_FIR_FORMAT_ANSI378 4

BSTR FIRTextData

Contains text encoded fingerprint data from the **Capture**, **Enroll**, and **CreateTemplate** methods. **FIRTextData** has variable size.

- Remarks**

Read only

BSTR StoredFIRTextData

Contains previously enrolled template. If **StoredFIRTextData** is set when calling **Enroll()**, it is merged with new FIR, **FIRTextData**. If **StoredFIRTextData** is not used, it should be set to "".

- Remarks**

Write only

BOOL IsMatched

Indicates the result of a match after using either the **Verify** or **VerifyMatch** methods. If the resulting value is 1 (**SecuBSP_TRUE**), a successful match has been made.

SecuBSP_FALSE 0

SecuBSP_TRUE 1

- Remarks**

Read only

BOOL IsPayload

Verifies the presence of any payload value in the **FIRTextData**. Payload can be extracted only when the match is successful. It is used with the **Verify** and **VerifyMatch** methods.

- Remarks**

Read only

BSTR PayloadData

Determines the value when payload data is detected within the **FIRText Data**. A value of 1 for the **IsPayload** indicates that the payload exists.

- Remarks**

Read only

LONG PayloadLength

Provides the length of the PayloadData

- **Remarks**
Read only

LONG MaxFingersForEnroll

Maximum number of fingers that can be enrolled into one template (ranges from 1 to 10, with default of 10)

LONG WindowStyle

Specifies the window style of the enrollment or verification/capture wizard

WS_CAPTURE_DEFAULT: Default window style is used in verification/capture. Default is a pop-up window.

WS_CAPTURE_INVISIBLE: Capture window does not show

WS_ENROLL_DEFAULT: Default window style is used in enrollment. Default is a pop-up window.

ENROLL_NO_WELCOMEPAGE: Does not show the welcome page in enrollment

A.2. Method

EnumerateDevice (void)

Provides the number and types of readers linked to the PC when used in conjunction with the DeviceNum and DeviceID properties

- **Remarks**
Related Properties: DeviceNum, DeviceID

OpenDevice (LONG deviceID)

Selects reader to use

- **Parameter**
DeviceID: sets the device ID to open
- **Remarks**
Related Property: DeviceID

CloseDevice (LONG deviceID)

Terminates usage of current selected reader

- **Parameter**
DeviceID: device ID to close. The DeviceID must be the same as the one previously used in the OpenDevice Method.
- **Remarks**
Related Property: DeviceID

MonitorDevice (BOOL enableMonitor, LONG hwnd)

Enables or disables the Auto-On function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. (Not supported by FDP0x or FDU02-based readers.)

- **Parameter**
enableMonitor: True: enable device monitoring function
False: disable device monitoring function
hwnd: Window handle to receive message from the device.
- **Remarks**
Related Property: DeviceID

AdjustDevice (void)

Controls the brightness of the captured image. Affects only open readers.

Capture (void)

Used to input one fingerprint and bring that information to the **FIRTextData** property

- **Remarks**

Related Property: FIRTextData

VerifyMatch (BSTR capturedTextFIR, BSTR storedTextFIR)

Performs matching using data from 2 FIRs. IsMatched shows the result of the match. Uses the IsPayload and PayloadData properties to bring forth any payload data from the FIRData.

- **Parameter**

capturedTextFIR: Fingerprint data captured for verification

storedTextFIR: Fingerprint data captured for enrollment

- **Remarks**

Related Property: IsMatched, IsPayload, PayloadData

CreateTemplate (VARIANT capturedTextFIR, VARIANT oldTextFIR, VARIANT vPayload)

Creates new FIR data by combining the old FIR Data or payload value. The new fingerprint data will be stored in the FIRTextData property.

- **Parameter**

capturedTextFIR: Newly captured fingerprint data

oldTextFIR: Existing fingerprint template

vPayload: Payload to be inserted into fingerprint data. Sets "Null" if there is no payload to be included.

- **Remarks**

Related Property: FIRTextData

Enroll (VARIANT vPayload)

Captures fingerprint data for enrollment. Up to 10 fingerprints can be enrolled. Captured fingerprint data is saved to FIRTextData property.

- **Parameter**

Payload to be inserted into fingerprint data. Sets "Null" if there is no payload to be included.

- **Remarks**

Related Property: FIRTextData

Verify (BSTR storedTextFIR)

Matches the captured fingerprint with storedTextFIR. The IsMatched property shows the result of the match. IsPayload indicates the presence of payload. The payload data can be found using the PayloadData property.

- **Parameter**

storedTextFIR: Data for the fingerprint originally registered

- **Remarks**

Related Property: IsMatched, IsPayload, PayloadData

SetSkinResource (BSTR resFileName)

Loads external resource file

- **Parameter**

resFileName: path of resource files. Should be the whole path.

Appendix B. SecuBSP .NET Reference

This chapter explains the SecuBSPMx class and other classes and the enumeration SecuBSP Pro .NET library (SecuBSPMx.NET.DLL). All classes and enumerations in the SecuBSP Pro .NET library are in the SecuGen.SecuBSPSDK.Windows namespace.

B.1. SecuBSPMx class

The SecuBSPMx class is the main class of the SecuBSP Pro .NET library. All SDK functions such as fingerprint image capture, extraction and matching are implemented in the SecuBSPMx class.

B.1.1. Constructor

SecuBSPMx()

SecuBSP class constructor

B.1.2. Properties

String* BSPVersion

Gets SecuBSPMx.DLL version

- **Remarks**
Read only

Int32 DeviceNum

Indicates the number of readers linked to the PC. This must be called after the **EnumerateDevice()**.

- **Remarks**
Read only

Int16 DeviceID

Device ID to be used in calling device related functions, **OpenDevice()**, **CloseDevice()** and **AdjustDevice()**.

DeviceID is composed of the device names and their instance numbers and has a 2 byte-length. The lower byte represents the device name, and the higher byte represents its instance number. **FDP01**- and **FDP02**-based (parallel) fingerprint readers have **0x01** as the device name. **FDU02**-based (USB) fingerprint readers have **0x02** as the device name. **FDU03**- and **SDU03**-based (USB) fingerprint readers have **0x03** as the device name. **FDU04**-based (USB) fingerprint readers have **0x04** as the device name. The instance number starts from 0. If there is only one reader for each type in the system, the instance number will be 0. In this way, the device name has the same value as the device ID. For example, **DeviceID** for the first instance of an FDP020-based reader will be 0x0001, and the DeviceID for the first instance of an FDU02-based reader will be 0x0002.

DeviceID = Instance Number + Device Name

Device Names

Device Name	Value	Notes
DeviceName.FDP02	0x01	FDP01 & FDP02 parallel readers
DeviceName.FDU02	0x02	FDU02 USB readers
DeviceName.FDU03	0x03	FDU03 & SDU03 USB readers
DeviceName.FDU04	0x04	FDU04 USB readers

Predefined Device IDs Already Declared

Device ID	Value	Notes
DeviceID.UNKNOWN	0(0x0000)	No device
DeviceID.PARALLEL_0	1(0x0001)	The first instance of FDP01 or FDP02
DeviceID.USB_0	2(0x0002)	The first instance of FDU02
DeviceID.USB_1	3(0x0003)	The first instance of FDU03 or SDU03
DeviceID.AUTO	255(0x00FF)	Detect device automatically

String* FIRTextData

FIRTextData has latest capture FIR data with text encoded type from the **Capture()**, **Enroll()**, and **CreateTemplate()** methods. **FIRTextData** has variable size.

- **Remarks**
Read only

FIRInformation* FIRInfo

FIRInfo contains FIR information of latest captured FIR from the **Capture()**, **Enroll()**, and **CreateTemplate()** methods. For more information about the FIRInformation class, see [Appendix B.6. FIRInformation class](#).

- **Remarks**
Read only

BOOL IsMatched

Indicates the result of the matching after using either the **Verify()** or **VerifyMatch()** methods. If a successful match has been made, it would be **true**.

true: Matched
false: Not matched

- **Remarks**
Read only

BOOL IsWideChar

Specifies whether or not **FIRTextData** has multi-byte format (default is **false**)

true: FIR text is wide character
false: FIR text is not wide character

BOOL IsPayload

Verifies the presence of any Payload value in the **FIRTextData**. Payload can be extracted only when the matching is successful. **IsPayload** should be checked after calling **Verify()** and **VerifyMatch()**.

true: FIR has payload
false: FIR does not have payload

- **Remarks**
Read only

String* PayloadData

Determines the value when payload data is detected within the FIRTextData. It is valid only when **IsPayload** is **true**.

- **Remarks**
Read only

WindowOption* EnrollWindowOption

Sets window option in the enrollment wizard. For more information, see the **WindowOption** class.

WindowOption* CaptureWindowOption

Sets window option in the verification/capture wizard. For more information, see the **WindowOption** class.

B.1.3. Methods**BSPErrors GetInitInfo(BSPInitInfo* initInfo);**

Gets initial status of the SecuBSP module

- **Parameters**
initInfo: Contains initial status of SecuBSP module. See also **BSPInitInfo** class
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration

BSPErrors SetInitInfo(BSPInitInfo* initInfo);

Sets initial status of the SecuBSP module

- **Parameters**
initInfo: See also **BSPInitInfo** class
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration

BSPErrors EnumerateDevice ()

Provides the number and types of readers linked to the PC when used in conjunction with **DeviceNum** and **GetDeviceID**

- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
Related members: **DeviceNum**, **GetDeviceID()**

Int16 GetDeviceID (Int32 index)

Used to get **DeviceID** of each reader after **EnumerateDevice()** method. After **EnumerateDevice()**, **DeviceNum** property has the number of readers linked to the PC. For each reader, if you pass device number (index) sequentially, it will return **DeviceID** of each reader.

- **Parameters**
index: Specifies device number. It should be between 0 and **DeviceNum - 1**
- **Return Value**
Device ID. **DeviceID** has a 2byte-length. The lower byte represents the Device Name, and the higher byte represents its instance number. **FDP01**- and **FDP02**-based (parallel) fingerprint readers have **0x01** as the device name. **FDU02**-based (USB) fingerprint readers have **0x02** as the device name. **FDU03**- and **SDU03**-

based (USB) fingerprint readers have **0x03** as the device name. **FDU04**-based (USB) fingerprint readers have **0x04** as the device name. The instance number starts from 0. If there is only one reader for each type in the system, the instance number will be '0.' In this way, the device name has the same value as the device ID. For example, the DeviceID for the first instance of an FDP02-based reader will be 0x0001, and the DeviceID for the first instance of an FDU02-based reader will be 0x0002.

$$\text{DeviceID} = \text{Instance Number} + \text{Device Name}$$

Device Name

Device Name	Value	Notes
DeviceName.FDP02	0x01	FDP01 & FDP02 parallel readers
DeviceName.FDU02	0x02	FDU02 USB readers
DeviceName.FDU03	0x03	FDU03 & SDU03 USB readers
DeviceName.FDU04	0x04	FDU04 USB readers

Predefined Device Ids

Device ID	Value	Notes
DeviceID.UNKNOWN	0(0x0000)	No devices
DeviceID.PARALLEL_0	1(0x0001)	The first instance of FDP01 or FDP02
DeviceID.USB_0	2(0x0002)	The first instance of FDU02
DeviceID.USB_1	3(0x0003)	The first instance of FDU03 or SDU03
DeviceID.AUTO	255(0x00FF)	Detect device automatically

BSPErrors.OpenDevice ()

Opens the reader to use. To specify the reader to open, set DeviceID before calling this method.

- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
Related Property: DeviceID

BSPErrors.CloseDevice ()

Terminates usage of the currently selected reader

- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
Related Property: DeviceID

BSPErrors.GetDeviceInfo(DeviceInfo* deviceInfo)

Gets device information of the currently opened reader

- **Parameters**
deviceInfo: See **DeviceInfo** class
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
Related members: OpenDevice(), DeviceID

BSPErrors.SetDeviceInfo(DeviceInfo* deviceInfo)

Sets device information of the currently opened reader

- **Parameters**

deviceInfo: See **DeviceInfo** class

- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
Related members: **OpenDevice()**, **DeviceID**

BSPErrors.AdjustDevice()

Displays dialog for adjusting the brightness of the captured image. Affects only the currently selected reader.

- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration

BSPErrors.MonitorDevice(bool enable, IntPtr hwnd)

Enables or disables the Auto-On function that allows the reader to automatically detect the presence of a finger without requiring the user to prompt the system before receiving a fingerprint. (Not supported by FDP0x or FDU02-based readers.)

- **Parameters**
enable: True to enable device monitoring, otherwise **SecuAPI.FALSE**
hwnd: Window handle to receive message from the device
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration

BSPErrors.SetFIRFormat(FIRFormat format)

Sets the FIR Format. SecuBSP Pro supports two types of FIR Format: **STANDARDPRO** and **ANSI378**. **STANDARDPRO** is encrypted. **ANSI378** is the pure (unencrypted) ANSI-INCITS 378-2004 format.

FIRFormat

Format name	Description	Value
STANDARDPRO	Encrypted FIR	3
ANSI378	Pure ANSI-INCITS 378 Format	4

- **Parameters**
format: SecuAPI FIR Format: **STANDARDPRO** or **ANSI378**
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
To get ANSI378 format FIR, **SetFIRFormat()** should be called before calling the **Enroll()** or **Capture()** function.

BSPErrors.Enroll(String* payload)

Captures fingerprint data for enrollment. Up to 10 fingerprints can be enrolled. Captured fingerprint data is saved to **FIRTextData** property.

- **Parameters**
payload: Payload to be inserted into fingerprint data. Sets empty string ("") if there is no payload to be included.
- **Return Value**
If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
After calling this method, to get a newly enrolled data, use **FIRTextData** property
Related Property: **FIRTextData**, **FIRInfo**

BSPErrors.Enroll(String* payload, String* storedFIR)

Captures up to fingerprints for enrollment; captured fingerprint data is saved to the **FIRTextData** property.

- **Parameters**
payload: Payload to be inserted into fingerprint data. Sets empty string ("") if there is no payload to be included.
storedFIR: A FIR data to be adapted
- **Return Value**
 If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
 After calling this method, to get a newly enrolled data, use **FIRTextData** property
 Related Property: **FIRTextData**, **FIRInfo**

BSPErrors.Capture()

Used to input one fingerprint. Captured fingerprint data is saved to **FIRTextData** property.

- **Return Value**
 If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
 After calling this method, to get a newly captured data, use **FIRTextData**.
 Related Property: **FIRTextData**, **FIRInfo**

BSPErrors.Verify (String* storedTextFIR)

Matches the currently captured fingerprint with the storedTextFIR. The **IsMatched** property shows the result of the match. **IsPayload** indicates the presence of payload. The payload data can be found using the **PayloadData** property.

- **Parameters**
storedTextFIR: previously stored fingerprint FIR
- **Return Value**
 If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
 Related Property: **IsMatched**, **IsPayload**, **PayloadData**

BSPErrors.VerifyMatch (String* capturedTextFIR, String* storedTextFIR)

Performs matching using two FIR data. **IsMatched** shows the result of the matching. If **IsMatched** is true and **storedTextFIR** has payload, then **IsPayload** will be true, and payload data can be accessed through **PayloadData**.

- **Parameters**
capturedTextFIR: Fingerprint data captured for verification
storedTextFIR: Fingerprint data captured for enrollment
- **Return Value**
 If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**
 Related Property: **IsMatched**, **IsPayload**, **PayloadData**

BSPErrors.CreateTemplate (String* capturedTextFIR, String* storedTextFIR, String* payloaddata)

Creates new FIR data by combining the old FIR data or payload data. The new fingerprint data will be stored in the **FIRTextData** property.

- **Parameters**
capturedTextFIR: Newly captured fingerprint data
storedTextFIR: Existing fingerprint template
payloaddata: Payload to be inserted into fingerprint data. Sets "Null" if there is no payload to be included.
- **Return Value**
 If function is successful, returns **BSPErrors.ERROR_NONE**. For error values, see **BSPErrors** Enumeration
- **Remarks**

Related Property: FIRTextData

bool SetSkinResource (String* resFileName)

Loads external customized SecuBSP UI resource file

- **Parameters**
resFileName: path of resource files. Should be the whole path.
- **Return Value**
Returns **true** if function was successful, otherwise false.

BSPError ImportFDxData(Byte fdxData[], FdxMinType fdxType, FIRPurpose purpose)

Imports 400 byte minutiae data created from FDX SDK or FDX SDK *Pro* and converts to SecuBSP FIR format. Converted data is contained in **FIRTextData**.

- **Parameters**
fdxData: The FDX data to be imported
fdxMinType: FDX data type.
FDxMinType.MIN_TYPE_SG400 (11): Minutiae data from FDX SDK *Pro*
FDxMinType.MIN_TYPE_OLD_FDX (12): Minutiae data from FDX SDK
purpose: The Purpose of FIR to be newly created
- **Return Value**
If function is successful, returns **BSPError.ERROR_NONE**. For error values, see **BSPError** Enumeration
- **Remarks**
Related Property: FIRTextData
Minutiae data from stand-alone type device (FDA or SDA) should use MIN_TYPE_SG400.

BSPError ExportFDxData(String* textFIR, FdxMinType fdxMinType)

Exports SecuBSP FIR format data into 400 byte minutiae data (FDX format). Converted data is contained in **ExportMinutiaeDataStruct**.

- **Parameters**
textFIR: The FIR data to be exported(or converted)
fdxMinType: FDX data type.
FDxMinType.MIN_TYPE_SG400 (11): Minutiae data from FDX SDK *Pro* or Stand-alone device
FDxMinType.MIN_TYPE_OLD_FDX (12): Minutiae data from FDX SDK
- **Return Value**
If function is successful, returns **BSPError.ERROR_NONE**. For error values, see **BSPError** Enumeration
- **Remarks**
Related Property: ExportMinutiaeDataStruct

BSPError ExportAuditData(String* auditTextFIR)

Exports image data from FIR audit data. Exported data information and image data are contained in **ExportImageDataStruct**.

- **Parameters**
auditTextFIR: The Audit FIR data to be exported
- **Return Value**
If function is successful, returns **BSPError.ERROR_NONE**. For error values, see **BSPError** Enumeration
- **Remarks**
Related Property: ExportImageDataStruct

DeviceName GetDeviceName(Int16 deviceId)

Gets device name from deviceId

- **Parameters**

deviceId: device ID

- **Return Value**

DeviceName.FDP02, DeviceName.FDU02, DeviceName.FDU03 or DeviceName.FDU04. If *deviceId* is wrong, then DeviceName.Unknown will be returned.

Byte GetDeviceInstanceNum(Int16 deviceId)

Gets device instance number from *deviceId*

- **Parameters**

deviceId: device id

- **Return Value**

Device instance number starting from 0

B.2. BSPInitInfo class

BSPInitInfo is used to get or set initial status of the SecuBSP module.

B.2.1. Constructor

BSPInitInfo()

BSPInitInfo class constructor

B.2.2. Fields

Int32 MaxFingersForEnroll

Maximum fingers for enrollment (ranges from 1 to 10, with default of 10)

Int32 SamplesPerFinger

Samples per finger. Default is 2. For verification purposes, 1 sample is captured. In the current version, this value is fixed at 2 for enrollment and 1 for verification.

Int32 DefaultTimeout

Default timeout value in milliseconds; used when capturing fingerprint image data from the reader. Default is 10000 milliseconds.

Int32 EnrollImageQuality

FIR quality on enrollment time (ranges from 30 (lowest) to 100 (highest), with default of 50)

Int32 VerifyImageQuality

FIR quality on verification time (ranges from 0 (lowest) to 100 (highest), with default of 30)

Int32 IdentifyImageQuality

FIR quality on identification time (ranges from 0 (lowest) to 100 (highest), with default of 50). Not used in current version.

Int32 SecurityLevel

Security level for verification (ranges from **SecurityLevel.Lowest** to **SecurityLevel.Highest**, with default of **SecurityLevel.Normal**). As the security level is increased, the FRR increases while FAR decreases. For more information, see SecurityLevel enumeration.

SecurityLevel.LOWEST

SecurityLevel.LOWER

SecurityLevel.LOW

SecurityLevel.BELOW_NORMAL
SecurityLevel.NORMAL
SecurityLevel.ABOVE_NORMAL
SecurityLevel.HIGH
SecurityLevel.HIGHER
SecurityLevel.HIGHEST

B.3. DeviceInfo class

B.3.1. Constructor

DeviceInfo()

DeviceInfo class constructor

B.3.2. Fields

Int32 ImageWidth

Image width in pixels (depends on reader type)

Int32 ImageHeight

Image height in pixels (depends on reader type)

Int32 Brightness

Brightness value of the reader (ranges from 0 to 100)

Int32 Gain

Gain value of the reader (depends on reader type)

Int32 ImageDPI

Image resolution in DPI

Int32 FWVersion

Firmware Version

Byte DeviceSN[]

Device serial number

B.4. ExportImageDataStruct class

B.4.1. Constructor

ExportImageDataStruct()

FIRInformation class constructor

B.4.2. Fields

Byte NumOfFingers

Number of fingers in input FIR

Byte SamplePerFinger

Number of samples per finger

Int32 ImageWidth

Image width

Int32 ImageHeight

Image height

FINGER_DATA_STRUCT ImageData[]

Array of each finger. Each finger has image data. See FINGER_DATA_STRUCT.

B.4.3. FINGER_DATA_STRUCT**Byte FingerPos**

Represents finger position

FingerID.UNKNOWN (0): Finger position in case of verification
 FingerID.RIGHT_THUMB (1)
 FingerID.RIGHT_INDEX (2)
 FingerID.RIGHT_MIDDLE (3)
 FingerID.RIGHT_RING (4)
 FingerID.RIGHT_LITTLE (5)
 FingerID.LEFT_THUMB (6)
 FingerID.LEFT_INDEX (7)
 FingerID.LEFT_MIDDLE (8)
 FingerID.LEFT_RING (9)
 FingerID.LEFT_LITTLE (10)

Byte Sample1[]

First sample data in FingerPos. It has 400 bytes minutiae data.

Byte Sample2[]

Second sample data in FingerPos. It has 400 bytes minutiae data. If number of samples (**SamplesPerFinger**) is 1, this field is not used.

B.5. ExportMinutiaeStruct class**B.5.1. Constructor****ExportMinutiaeDataStruct()**

FIRInformation class constructor

B.5.2. Fields**Byte NumOfFingers**

Number of fingers in input FIR

Byte SamplePerFinger

Number of samples per finger

FINGER_DATA_STRUCT MinutiaeData[]

Array of each finger. Each finger has minutiae data. See FINGER_DATA_STRUCT.

B.5.3. FINGER_DATA_STRUCT

Byte FingerPos

Represents finger position

FingerID.UNKNOWN (0): Finger position in case of verification
 FingerID.RIGHT_THUMB (1)
 FingerID.RIGHT_INDEX (2)
 FingerID.RIGHT_MIDDLE (3)
 FingerID.RIGHT_RING (4)
 FingerID.RIGHT_LITTLE (5)
 FingerID.LEFT_THUMB (6)
 FingerID.LEFT_INDEX (7)
 FingerID.LEFT_MIDDLE (8)
 FingerID.LEFT_RING (9)
 FingerID.LEFT_LITTLE (10)

Byte Sample1[]

First sample data in FingerPos. The contained image data size is ImageWidth*ImageHeight.

Byte Sample2[]

Second sample data in FingerPos. The contained image data size is ImageWidth*ImageHeight. If number of samples (**SamplesPerFinger**) is 1, this field is not used.

B.6. FIRInformation class

B.6.1. Constructor

FIRInformation()

FIRInformation class constructor

B.6.2. Fields

Int32 Format

Indicates the format for FIR data as either Standard format or SecuIBAS extended format. The SecuBSP SDK supports only the STANDARD format. Contains one of the following values. For more information, see **FIRFormat** enumeration.

FIRFormat.STANDARD
 FIRFormat.SECUIBAS

Int16 Version

FIR version number. Current version is 2.

Int16 DataType

The type of fingerprint data stored in the FIR. Contains one of the following values. For more information, see **FIRDataType** enumeration.

FIRDataType.RAW
 FIRDataType.INTERMEDIATE
 FIRDataType.PROCESSED
 FIRDataType.ENCRYPTED

Int16 Purpose

The purpose of the FIR. Contains one of the following values. For more information, see **FIRPurpose** enumeration.

```
FIRPurpose.VERIFY
FIRPurpose.IDENTIFY
FIRPurpose.ENROLL
FIRPurpose.ENROLL_FOR_VERIFICATION_ONLY
FIRPurpose.ENROLL_FOR_IDENTIFICATION_ONLY
FIRPurpose.AUDIT
FIRPurpose.UPDATE
```

Int16 Quality

Indicates the quality of the fingerprint data (ranges from 1 (lowest) to 100 ((highest))

B.7. WindowOption class

B.7.1. Property

Int32 WindowStyle

Sets window style of the verification /capture wizard. Can be one of the following values. For more information, see **WindowStyle** enumeration.

```
WindowStyle.POPUP: Default window
WindowStyle.INVISIBLE: Hide window
```

Int32 FingerWindow

Sets window for displaying fingerprint image

Int32 ParentWindow

Sets parent window of the enrollment wizard or verification wizard

bool WelcomePage

Shows or hides welcome page in the enrollment wizard

bool ShowFPImage

Shows or hides fingerprint image in the verification/capture wizard

B.8. Constants and Enumeration

B.8.1. BSPError Enumeration

Represents SecuBSP error value

Members

Member name	Description	Value
ERROR_NONE	Function completed successfully	0
ERROR_INVALID_HANDLE	Invalid handle in an input function parameter or input field of a data structure	1
ERROR_INVALID_POINTER	Invalid pointer in an input function parameter or input field of a data structure	2

ERROR_INVALID_TYPE	Input structure type is invalid	3
ERROR_FUNCTION_FAIL	Function failed for unknown reason	4
ERROR_STRUCTTYPE_NOT_MATCHED	Type of input structure does not match request type value	5
ERROR_ALREADY_PROCESSED	Template was already processed	6
ERROR_EXTRACTION_OPEN_FAIL	Extraction module cannot be opened	7
ERROR_VERIFICATION_OPEN_FAIL	Verification module cannot be opened	8
ERROR_DATA_PROCESS_FAIL	Internal error occurred during data processing	9
ERROR_MUST_BE_PROCESSED_DATA	Function requires a fully processed FIR	10
ERROR_INTERNAL_CHECKSUM_FAIL	Checksum of FIR data in input parameter is invalid	11
ERROR_ENCRYPTED_DATA_ERROR	Encrypted FIR data in input parameter is broken	12
ERROR_UNKNOWN_FORMAT	Input parameter contains unknown FIR data format	13
ERROR_UNKNOWN_VERSION	Input parameter contains unknown FIR data version	14
ERROR_VALIDITY_FAIL	Not currently used	15
ERROR_INIT_MAXFINGER	Maximum finger count for enrollment is invalid (ranges from 1~10)	16
ERROR_INIT_SAMPLESPERFINGER	Samples-per-finger count for enrollment is invalid (ranges from 1~10)	17
ERROR_INIT_ENROLLQUALITY	Image quality value for enrollment is invalid (ranges from 0~100)	18
ERROR_INIT_VERIFYQUALITY	Image quality value for verification is invalid (ranges from 1~100)	19
ERROR_INIT_IDENTIFYQUALITY	Image quality value for identification is invalid (ranges from 0~100)	20
ERROR_INIT_SECURITYLEVEL	Security level value is invalid	21
ERROR_INVALID_MINSIZE	Template size is invalid	22
ERROR_INVALID_TEMPLATE	Invalid template	23
ERROR_DEVICE_OPEN_FAIL	Reader could not be opened	257
ERROR_INVALID_DEVICE_ID	Invalid device ID	258
ERROR_WRONG_DEVICE_ID	Wrong device ID	259
ERROR_DEVICE_ALREADY_OPENED	Reader is already opened	260
ERROR_DEVICE_NOT_OPENED	Reader is not yet opened	261
ERROR_DEVICE_BRIGHTNESS	Brightness value is invalid (ranges from 0~100)	262
ERROR_DEVICE_CONTRAST	Contrast value is invalid (ranges from 0~100)	263
ERROR_DEVICE_GAIN	Gain value is invalid. Depends on the reader.	264
ERROR_DEVICE_INFO_FAILED	Failed to get device information	265
ERROR_USER_CANCEL	Use of "Cancel" button closed function	513
ERROR_USER_BACK	Use of the "Back" button closed function	514
ERROR_CAPTURE_TIMEOUT	Fingerprint capture process timed out and closed the function	515

B.8.2. DriverEvent Enumeration

Indicates device event

Members

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

B.8.3. DriverMessage Enumeration

Indicates messages from the device

Members

Members name	Description	Value
WM_DEVICE_EVENT	Message from the fingerprint reader	0x8100

B.8.4. FingerIDEnumeration

Indicates finger position in FIR

Members

Members name	Description	Value
UNKNOWN	Used for Verification	0
RIGHT_THUMB	Right thumb	1
RIGHT_INDEX	Right index finger	2
RIGHT_MIDDLE	Right middle finger	3
RIGHT_RING	Right ring finger	4
RIGHT_LITTLE	Right little finger	5
LEFT_THUMB	Left thumb	6
LEFT_INDEX	Left index finger	7
LEFT_MIDDLE	Left middle finger	8
LEFT_RING	Left ring finger	9
LEFT_LITTLE	Left little finger	10

B.8.5. FDxMinType Enumeration

Indicates the minutiae type of fingerprint data

Members

Members name	Description	Value
MIN_TYPE_SG400	Minutiae data from FDx SDK Pro	11
MIN_TYPE_OLD_FDX	Minutiae data from FDx SDK	12

B.8.6. FIRDataType Enumeration

Indicates the type of fingerprint data stored in the FIR

Members

Members name	Description	Value
RAW	FIR data is raw	0x00
INTERMEDIATE	FIR data is intermediate	0x01
PROCESSED	FIR data is processed	0x02
ENCRYPTED	FIR data is encrypted	0x10

Remarks

FIR data after **Enroll()** or **Capture()** will have a PROCESSED data type.

B.8.7. FIRPurpose Enumeration

Indicates the purpose of FIR. When used with **Capture()**, specifies capture purpose.

Members

Member name	Description	Value
VERIFY	Verification purpose	0x01
IDENTIFY	Identification purpose	0x02
ENROLL	Enrollment purpose	0x03
ENROLL_FOR_VERIFICATION_ONLY	Enrollment for verification only purpose	0x04
ENROLL_FOR_IDENTIFICATION_ONLY	Enrollment for identification only purpose	0x05
AUDIT	Audit purpose	0x06
UPDATE	Update purpose	0x10

B.8.8. SecurityLevel Enumeration

Indicates the security level set for fingerprint recognition (ranges from 1 (lowest) to 9 (highest), with default of 5 (normal)). The **SecurityLevel** enumeration is used to set security level in **SetInitInfo()** method and is applied globally.

Members

Member name	Description	Value
LOWEST	Most lowest security level	1
LOWER	Lower security level	2
LOW	Low security level	3
BELOW_NORMAL	Below than normal security level	4
NORMAL	Normal security level. Default.	5
ABOVE_NORMAL	Above than normal security level	6
HIGH	High security level	7
HIGHER	Higher security level	8
HIGHEST	Most highest security level	9

B.8.9. WindowStyle Enumeration

Specifies window style in Verification/Capture window

Members

Member name	Description	Value
POPUP	Window style is pop-up.	0
INVISIBLE	Windows is not visible.	1

B.8.10. FIRFormat Enumeration

Indicates FIR format of fingerprint data

Members

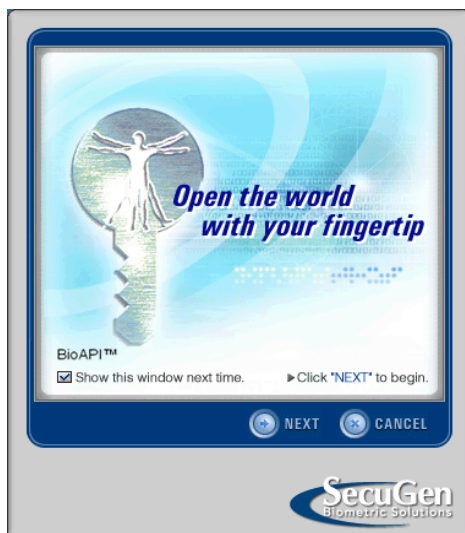
Member name	Description	Value
STANDARDPRO	Default FIR (encrypted)	3
ANSI378	Pure ANSI378 Format	4

Appendix C. Using the Wizards

SecuBSP contains graphical “wizards” that encapsulate sophisticated fingerprint algorithms and image capture technology in very simple and easy to use interfaces that may be presented to the user for fingerprint enrollment and verification.

C.1. Enrollment Wizard

1. When the SecuAPI “Enroll” function is called, the SecuBSP fingerprint registration wizard is displayed as shown below. Click **NEXT** to continue through the fingerprint registration process. The initial welcome screen is optional. If you do not want to show this screen the next time, uncheck the box “*Show this window next time.*”



2. When the enrollment process commences, two illustrated hands will appear in the next window, corresponding to the user's hands. Click on the circle above the finger you wish to register.



- After selecting a finger to enroll, the fingerprint capture window is displayed. Two fingerprint images must be captured from the same finger to complete registration



- Place finger on the fingerprint sensor when prompted.

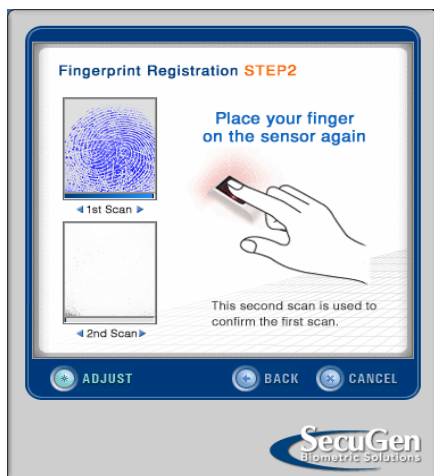


Note: You may click the "Adjust" button to adjust the brightness and contrast of the SecuGen reader if the fingerprint image is too dark, too light, or without enough contrast.

- When the first fingerprint image has been captured, remove the finger when prompted.



6. Place the **same** finger back on the sensor of the fingerprint reader when prompted. It is necessary to capture a second image of the same finger to complete enrollment.



7. After the second fingerprint has been successfully captured it will be displayed in the "Quality Check" window. During this process, minutiae data is extracted from the fingerprint image, and then all fingerprint image data is erased.



8. The circle above the registered finger is now colored light purple. If you want to register more than one finger, click another finger to enroll. To finish fingerprint registration, click **NEXT**. If you select another finger to enroll, the registration process described above will be repeated.

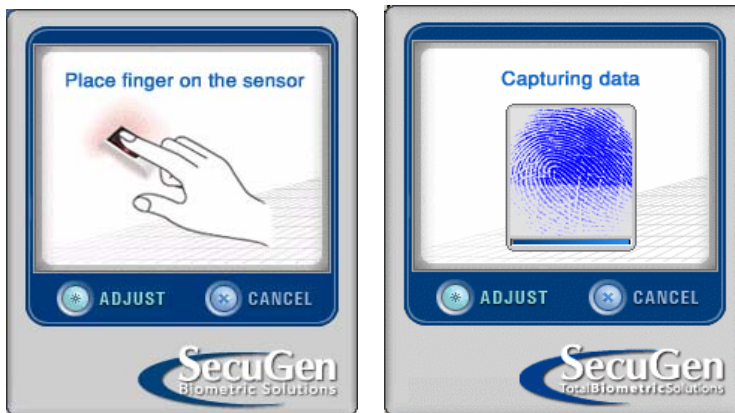


9. After the fingerprint registration process has been successfully completed, the dialog shown below will be displayed. To complete registration, click **FINISH**.



C.2. Verification Wizard

When the SecuAPI “Verify” function is called, the SecuBSP BSP fingerprint verification wizard is displayed. This wizard uses the same user-friendly interface as the fingerprint registration process. Just place the finger on the center of the fingerprint reader when prompted. Verification is successful if the user’s fingerprint minutiae score is matched to the enrolled fingerprint template.



Note: You may click the “Adjust” button to adjust the brightness and contrast of the SecuGen reader if the fingerprint image is too dark, too light, or without enough contrast.

C.3. Brightness adjustment wizard

For optimal performance it is important to capture good, clear images of the fingerprints for enrollment and verification.

1. If the fingerprint appears to be too dark or too light, click the **ADJUST** button in either the Registration Wizard or the Verification Wizard. The adjustment process requires a fingerprint to be present. Start by positioning your finger or thumb on the SecuGen fingerprint reader when prompted and hold it in place.



2. Now, with your finger in place, click and drag the adjustment bar to left (darker image) or right (brighter image). Your fingerprint image will display in the fingerprint image window. Keeping your finger in place on the sensor, click **DONE** when the image is clearly defined (i.e. fingerprint ridges and valleys should appear as bright and dark lines with enough contrast).