Interface Specification

Human Authentication - Application Program Interface (HA-API)

Ver 2.0

THE DEPARTMENT OF DEFENSE HAS FUNDED THIS SPECIFICATION AS A RESEARCH PROJECT AIMED AT INTEROPERABILITY AND EASE IN IMPLEMENTATION OF BIOMETRIC ACCESS CONTROLS FOR U.S. GOVERNMENT AND DOD COMPUTERS. IT DOES NOT REFLECT U.S. GOVERNMENT ENDORSEMENTS OF HA-API COMPLIANT PRODUCTS OR ENDORSEMENT OF THIS SPECIFICATION OVER OTHER INDUSTRY GENERATED BIOMETRIC API STANDARDS.

22 April 1998

Comments

Comments on the Human-Authentication API may be sent to:

Major John Colombi, Ph.D., USAF
U.S. Biometrics Consortium
c/o NSA/ R22, Suite 6516
9800 Savage Road
Fort Meade, MD 20755-6516

or emailed to:

jmcolom@alpha.ncsc.mil

Table of Contents

1.0 Scope 1

1.1 Purpose 1

1.2 Background 1

1.3 Overview 2
1.3.1 Identification & Authentication 2
1.3.2 Biometrics 3
1.3.3 Windows-NT 3
1.3.4 Enrollment 3
1.3.5 Matching 4
1.3.6 Philosophy 5

2.0 Architecture 8

3.0 Functions 11

3.1 Enrollment 11

3.2 Matching 12

4.0 API Definitions 13

4.1 Biometric Technology Functions 14
4.1.1 EnumBioTechnology 14
4.1.2 GetBioTechnology 15
4.1.3 ReleaseBioTechnology 15

4.2 Biometric Authentication Functions 16
4.2.1 HAAPICapture 16
4.2.2 HAAPIProcess 17
4.2.3 HAAPIVerify 18
4.2.4 HAAPILiveVerify 20
4.2.5 HAAPIEnroll 22
4.2.6 HAAPIUpdate 25
4.2.7 HAAPIIdentify 27

4.3 Biometric Utility Functions 27

4.3.1 HAAPIFree 27
4.3.2 HAAPIInformation 28
4.3.3 HAAPIBioProperties 30

5.0 Service Provider Interface 32

5.1 Description 32
5.1.1 BUID 33
5.1.2 Asynchronous Operation 34
5.1.3 Authentication of BSP module 34
5.1.4 ANSI vs. UNICODE 34

5.2 User Interface 34

5.3 The Enrollment Wizard 35
5.3.1 Exported Enrollment Function 35

5.4 Update Enrollment 39
5.4.1 Exported Update Function 39

5.5 Standard Capture Screen 40
5.5.1 Exported Capture Function 41

5.6 Processing Interface 42
5.6.1 Exported Process Function 42

5.7 Verify Interface 43
5.7.1 Exported Verify Function 43

5.8 LiveVerify Interface 45
5.8.1 Exported Live Verify Function 45

5.9 Free Memory 47
5.9.1 Exported Free Function 48

5.10 Technology Specific Parameters 48
5.10.1 Exported Information Function 48

5.11 Standard Biometric Properties Screen 50
5.10.1 Exported Biometric Properties Function 50

6.0 Structures 52

6.1 Raw Biometric Data Structure 52

6.2 Biometric Identifier Record Structure 53

6.3 HA-API Header Structure 54

6.4 Biometric Technology Structure 56

6.5 BUID Structure 56

6.6 Scoring Record Structure 57

6.7 Scoring Data Structure 58

6.8 Threshold Record Structure 59

6.9 Threshold Data Structure 60

6.10 Screen Attributes Structure 61

6.11 Enrollment Pages Structure 62

7.0 Sample message flows 63

7.1 Enrollment 63
7.1.1 New Enrollment 63
7.1.2 Update Enrollment 64
7.1.3 Batch Enrollment 65

7.2 Verification 66
7.2.1 Normal Verify 66
7.2.2 Live Verify 67
7.2.3 Verify with Update 68

8.0 References 69

9.0 Glossary 70

9.1 Acronyms 70

9.2 Definitions 71

Appendix A - Function Prototypes 72
Appendix B - Defines 74
Appendix C - Enumerated Types 75
Appendix D - C++ Classes 76
Appendix E - Error Codes, BUIDs and GUIDs 77
Appendix F - Scores and Thresholds 79
Appendix G - Sample Screens 81
Enrollment Capture Screen 81
Dialog Box 82
Message Box 82

REVISION HISTORY

Revision	Date of Issue	Revision Summary
1.0	27 Aug 97	Baseline
1.01	24 Oct 97	Unicode strings, HAAPI function names
1.02	21 Nov 97	Vendor independent
1.03	30 Dec 97	Incorporate mods from pilot implementation: Add function - HAAPIFree Add parameter - lpScreenAttribs Add parameter - lpPages Restructure error codes Break out Biometric Utility function as category Differentiate API from pilot implementation
1.04	2 Apr 98	Incorporate changes agreed at 22 Jan 98 HA-API Steering Group meeting: Add function - HAAPI Update Add function - HAAPIBioProperties Added Score/Threshold Structure and discussion Added support for model adaptation in verify functions Added parameters to several functions to add flexibility Added several clarifying notes Added GUID / Error Code Description Additional sample flows
2.0	22 Apr 98	Feedback from Steering Group - grammatical changes only

1.0 Scope

1.1 Purpose

The purpose of this document is to define a generic human authentication - application programming interface (HA-API) that can be used to interface computer software applications to a set of distinct biometric technologies for user authentication.

1.2 Background

There is interest in the Department of Defense in applying biometrics to the broad area of computer security. One specific area that biometrics can be utilized is user identification and authentication (I&A).

Providing strong user authentication has long been a dilemma for information systems security professionals. Many excellent mechanisms exist today to authenticate one end of a communications link or network to the other. However, these peer-entity authentication mechanisms fail to perform the final step – user authentication. Without strong user authentication, a user has little assurance that the intended recipient actually is at the other end of the communications path.

To accelerate the commercial development of applications that use biometrics for positive identification purposes, it was determined that a generic biometric API was required. Such an API would allow a common set of instructions to be used to integrate a wide range of biometric technologies to many different applications, including computer/network I&A.

To meet the goal of expanding the use of biometrics by allowing for the interchangeability of biometric technologies within a broad range of applications, a two part project was conceived:

1. Define a generic biometric API that can be used to interface a computer software application to a set of various biometric technologies

2. Demonstrate the viability of this API by extending a commercial product, using at least 2 different biometric technology engines, as a proof of concept.

A standard, generic API is needed within the biometric industry for a number of reasons.

One of the inhibitors to the widespread adoption of biometrics has been the hesitancy of system integrators to get "locked in" to a single biometric technology, vendor, or product. Some undergo extensive evaluations to be sure they pick the right biometric, and others adopt a "wait and see" approach. A standard API would allow an integrator to go forward with a tentative selection, programming to the API. Should he later decide that another biometric would be more suitable, it could be substituted with minimal changes to the calling application. In addition to allowing substitution of biometrics, a common API would also provide for leveraging of a single biometric technology across multiple applications as well as allowing one application to integrate multiple biometric technologies using the same interface.

A proof-of-concept was performed which implemented HA-API within a commercial network authentication product with multiple biometric technologies in a Windows-NT environment. This implementation of the HA-API interface successfully completed beta testing in January 1998 and successfully demonstrated the viability of the API.

The HA-API was initially published for public comment on 21 Nov 97 as Ver 1.02. On 22 Jan 98, a HA-API Steering Group was formed and met to discuss comments, enhancements, and support for HA-API as an emerging standard. As a result of the inputs compiled at this meeting, this current version was generated.

1.3 Overview

1.3.1 Identification & Authentication

One of the basic methods used to maintain the security of a computer or network system is to verify the identity of the user – ensuring that the user is who he or she claims to be. Reliable authentication mechanisms are critical to the security of any automated information system as well as other access control systems. If the identity of legitimate users can be verified with an acceptable degree of accuracy, those attempting to gain access without proper authorization can be denied permission to use the system. Once verified, access control techniques can be applied to mediate that user's access to specific system resources. In addition, the user's actions while using the system can be audited.

There are three (3) generally accepted methods for performing user authentication. These are based on:

a. Something the user KNOWS (such as a *password*)

b. Something the user POSSESSES (such as a card/badge, called *tokens*)

c. Something the user IS (a physical characteristic, or *biometric*, such as a fingerprint)

These may be used independently or in conjunction with one another, to further increase the security level of the system.

Passwords. This is the most commonly used authentication mechanism today. However, passwords can be compromised in many ways - they can be forgotten, written down, guessed, stolen, "cracked", or shared.

Tokens. The identity of a user can be proven by requiring that the user demonstrate possession of a physical object that is unique to that user, or group of users. These are usually encoded with information used in the authentication process. Tokens can be lost, forgotten, stolen, given away, or duplicated.

Biometrics. Authentication can be accomplished by measurement of a unique biological or behavioral feature of the user to verify identity through automated means.

1.3.2 Biometrics

Biometric identification exploits the universally recognized fact that certain biological or behavioral characteristics reliably distinguish one person from another. Some examples of these characteristics are speech patterns, DNA, retinal patterns, the topography of the face, and the patterns of friction ridges on an individual's fingertip. A biometric characteristic is tightly bound to an individual. It can't be lost, forgotten, borrowed, stolen and duplicated, or otherwise compromised in the same manner as with ID cards, passwords, or PINs.

1.3.3 Windows-NT

Windows-NT is a graphical user interface (GUI) based computer operating system developed by Microsoft Corporation primarily for use on IBM-compatible PCs and servers. It supports a networked, client/server environment. It contains many security features not available in other commercially available operating systems. The Windows NT 3.51 release has been certified to the C2 level of security, requiring minimum user log-in procedures, auditing of security-relevant events, and resource (users, processes, and data) isolation. (See DoD 5200.28-STD, under Section 8.0, References, for further information on C2 security requirements.)

1.3.4 Enrollment

Before a biometric can be used to verify an identity, the user must be "enrolled" in the system. That is, their biometric identifier or template, along with their user ID, must be entered into the database of authorized users. A system/security administrator normally performs this function in order to protect the integrity of the authentication database.

In a password protected system, the system administrator or system security administrator (SA/SSA) would either allow the user to select or would assign the user ID and password. In a biometric authentication system, the biometric identification data for the user must be captured and stored. This entails capturing the raw biometric data, converting it to a biometric identifier or template, and storing it. For example:

> For a finger imaging system, one or more fingerprints are scanned one or more times using a finger image scanner device and the resulting digital fingerprint image is used to generate a Finger Image Identifier Record (FIIR).
>
> For a facial recognition system, all or part of the face is 'photographed' using a video or other type of photo camera. One or more images may be required. These images are used to generate the template(s), that may contain either extracted feature information or digital image data.
>
> For a speaker verification system, samples of the users speech, usually repeating predetermined phrases, are captured using a microphone or telephone handset. Speech signature parameters are calculated.

Enrolled biometric identifier data is stored in a protected authentication database. In a networked environment, this is usually located at the primary domain controller (PDC) or other authentication server.

1.3.5 Matching

To determine if one biometric sample "matches" another biometric sample, they must be compared using a unique algorithm. Generally, the result of this comparison is a "score", indicating the degree to which a match exists. This score is then compared to a pre-set threshold to determine whether or not to declare a match. The comparison is performed using the biometric identifier or template, as opposed to the raw biometric data that is captured.

Two types of matching are generally defined. These are 'Verification' and 'Identification'.

Verification is a one-to-one (1:1) matching of a single biometric sample set (biometric identifier record) against another. Generally, the first sample is newly captured and the second is the enrolled identifier on file for a particular subject. The file sample is retrieved from the database based on a unique subject identifier (such as a User ID). In a user authentication environment, a score exceeding the threshold would return a 'match', resulting in the authentication of the user. A score below the threshold would return a 'no-match', resulting in the denial of access.

Identification is a one-to-many (1:N) matching of a single biometric sample set against a database of samples, with no declared identity required. The single biometric is generally the newly captured sample and the database contains all previously enrolled samples. Scores are generated for each comparison, and an algorithm is used to determine the matching record, if any. Generally, the highest score exceeding the threshold results in a match. In an authentication environment, if a match is found against any of the authorized users, access is granted.

A third type, sometimes referred to as 'one-to-few' matching is performed by executing a series of 1:1 matches against a small sample set. In a user authentication environment using finger imaging technology, the user's finger image identifier is compared to each of the few authorized users within this group. Access is granted if any of the one-to-one matches is positive.

1.3.6 Philosophy

The approach herein adopted for the human authentication API is to hide to the degree possible, the unique aspects of individual biometric types and particular vendor implementations, products, and devices, while providing a 'toolbox' of biometric functions that can be used within a number of potential software applications. Access to this toolbox would be through a set of standard interfaces. Theoretically, biometric components supplied by vendors conforming to this interface specification could then be used within any application also developed to this HA-API definition.

This API is designed for use by both the application developer and the biometric technology developer. To make the integration of the technology as straight-forward and simple as possible (and thus enhancing its commercial viability), the approach taken was to hide or encapsulate to the extent possible the complexities of the biometric technology. This approach also serves to extend the generality of the interface to address a larger set of potential biometric technologies and applications thereof.

A broad range exists as to the level of detail to which such an interface specification could be defined. This range extends from the top-level functions of "Enroll" and "Verify" to the lowest level subfunction involving device manipulations. A level below the top-level functions was selected as being the most useful while remaining the most generic. It is believed that this level will provide the needed commonality without reducing all biometric technologies to the lowest common denominator. That is, without neutralizing competitive features offered by a vendor or product.

This specification is designed to support multiple biometrics, both singularly and when used in a combined or cascaded manner. This is desirable to enhance flexibility to support such implementations as multiple factor authentication.

For this version of HA-API, only one-to-one verification type matching is supported by the biometric API (and thus, so is one-to-few). One-to-many identification matching functions are not generally required in an authentication environment and pose additional challenges and considerations (such as accuracy, response time, adjudication of multiple candidates, and database synchronization issues) that would detract from the current effort. However, the specification was written so as not to preclude the addition of one-to-many identification matching in the future.

The issue of matching scores, threshold settings, and quality scores/thresholds are at this point vendor specific and generally not required for most 1:1 applications and integrators. However, there are some sophisticated developers and some unique applications where this functionality would be useful. Therefore, within the HAAPIInformation function, explicit variables have been defined to allow the setting and reading of these values. Note that the interpretation of these values continue to be vendor specific.

In addition to supporting operational system applications, this API could also be used within non-operational environments (e.g., to support biometric testing) as well. However, the Information function described above would likely be invoked to provide for the condition setting and detailed results usually necessary for any type of rigorous testing. Vendors are encouraged to provide a test mode or toolkit either separately or as an optional function.

In defining the approach, the physical location where the verification function takes place and the location of the identifier database is also a consideration. This API supports either local (workstation) or central (server) verification. To be most generic and to support a client/server environment, it is recommended that the biometric database be located at the central server. This is the cleanest implementation and supports either local or server verification.

As each system and application will employ different database types and designs, database related functions (such as the addition, deletion, and retrieval of biometric data) have thus been omitted from this API specification.

It is the intent that this API be as open as possible, both in terms of an open systems architecture and accommodation of as wide a range as possible of biometric products and vendors and a broad range of biometric applications. It is not intended to be platform or device dependent. For convenience, and to support the first prototype implementation, the software environment is currently defined in terms of a Win32 client/server environment using a "C" language based.

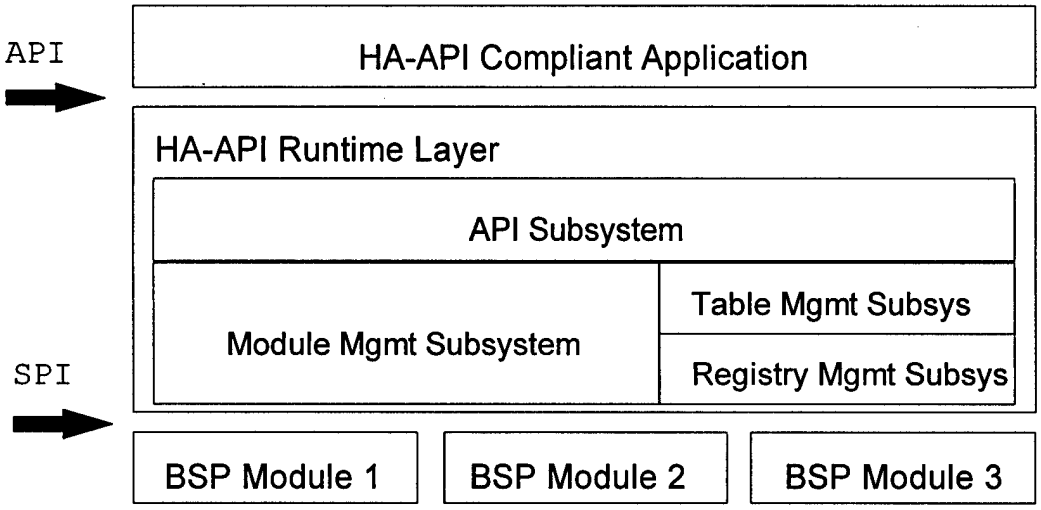
Use of encryption to protect biometric and other authentication data during transmission and storage is highly encouraged. No explicit cryptographic functions/parameters are included within this API for several reasons. First, since the application is responsible for data storage, it should be responsible for encrypt this data at all points where it may be vulnerable in the computer, when stored, and when transmitted. For data transmitted from

a biometric device or stored therein, this is the responsibility of the BSP vendor. Secondly, cryptographic API's already exist which can provide the necessary encryption capability.

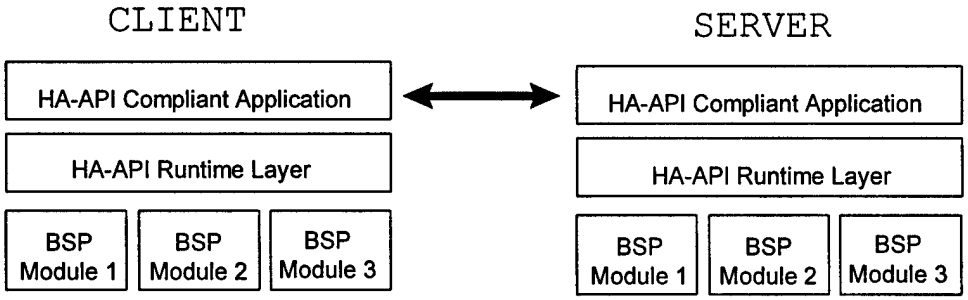
In order for an application to identify and distinguish between installed biometric technologies, each vendor/product pair is to be identified by a Biometric Unique Identifier (BUID). Vendors generate their own unique BUIDs using the Globally Unique Identifier (GUID) creation methods, which ensures that no two 128-bit values is generated twice. Note that once a BUID is generated for a vendor/product pair (i.e., a specific BSP), it is fixed for that product, although a vendor may choose to use a new BUID for a major revision to that product, at their discretion.

2.0 Architecture

This HA-API specification defines interfaces at two levels. The first is the application program interface (API). These functions are called by the software application that will use the biometrics for a particular purpose. The second is the service provider interface (SPI). These functions are provided by the biometric service provider (BSP). Translation from the SPI to the API is provided by the HA-API runtime layer. This is depicted in the figure below.



The HA-API runtime is free software currently available from NRI. Alternatively, the biometric technology vendor or the system integrator may develop it, if desired. When used in a client/server architecture, the implementation would be accomplished as shown below:



The intended system architecture integrating HA-API includes the following:

- Standalone or client/server implementation
- Open System environment
- Local, central, or distributed biometric database
- Support for local or central matching
- Support for multiple biometrics (individually)
- Support for hybrid implementation (in combination)

The salient points of the HA-API architecture are:

(a) Application will use the HA-API interface functions to select/configure the specific biometric technology used for user authentication process.

(b) The HA-API interface will include API functions to configure the specific required biometric capture device and the associated biometric engine for the biometric authentication process.

(c) The low level device driver software for communicating with the biometric devices and the biometric engine software will be part of the software package provided by the specific biometric vendor.

(d) Application will be responsible for maintaining the specific biometric Identifier database along with User ID related information in the NT Database for the user authentication process.

The primary design objective is to provide multiple biometric based user authentication. This design, will also provide the following operational advantages/enhancements:

- Enhance Interoperability
  - Flexibility to upgrade and/or switch biometric technology keeping the same application.
  - Flexibility to upgrade and/or switch biometric vendor (for a specific biometric technology) keeping the same application.

- Enhance Security /Reduce Risk
  - Ability to add /combine multiple biometric technologies to provide increased security.
  - Use of multiple combined biometric technologies to make the system work for universal population and also reduce false accept/false reject errors.

- Standard Biometric human authentication-API interface.
  - HA-API will facilitate the integration of a multitude of biometric technologies for user authentication.
  - Application will have the flexibility to add/switch/or combine biometric technologies easily without requiring a major software redesign.

3.0 Functions

The two top-level biometric functions for an authentication or other application are enrollment and verification matching, as described in previous sections. To perform these top-level function, subfunctions are defined that together, when called as API elements, execute these functions. The subfunctions are listed below. API specifications are provided in Section 4.0.

3.1 Enrollment

The following functions are used to perform the biometric enrollment function:

Initialize/release biometric

EnumBioTechnology
GetBioTechnology
ReleaseBioTechnology

Enroll biometric data

HAAPIEnroll
and optionally
HAAPIUpdate

Store biometric data

Application dependent

Get or set technology data/parameters (as needed)

HAAPIInformation

Memory De-allocation

HAAPIFree

3.2 Matching

The following functions are used to perform the biometric matching function. Only verification (1:1 matching) is supported by this version of the HA-API.

Initialize biometric, if not previously performed:

EnumBioTechnology
GetBioTechnology
ReleaseBioTechnology

Capture biometric data:

HAAPICapture

Calculate biometric template:

HAAPIProcess

Perform verification:

HAAPIVerify

or

HAAPILiveVerify

Get or set technology data/parameters (as needed)

HAAPIInformation

Memory De-allocation

HAAPIFree

4.0 API Definitions

The philosophy underlying Human Authentication-API (HA-API) interface design is to define the minimum set of generic API functions that will be required for providing multiple 1:1 biometric authentication capability for a HA-API compliant application. This interface design assumes that all of the low level functions (including the initialization of the data capture hardware boards for the capture of the biometric input data) and the actual processing /matching functions are handled internally within the biometric vendors' software libraries and engines.

Based on this assumption, the HA-API interface needs to support the following basic biometric identification functions that are common to all biometric technologies:

(a) Capture of biometric input data from the user.

(b) Creation of biometric identifier data used for the authentication function.

(c) Matching of the user's captured biometric identifier data against the stored biometric identifier data for the biometric authentication function.

Storage of the biometric identifier data is considered an application function.

A set of API functions has been defined for the HA-API interface for supporting multiple biometric authentication capability for the user authentication process. The initial HA-API interface is designed to provide a generic API interface that can support a variety of biometric technologies. A detailed description of the proposed API functions is provided in the remainder of this section. The API interface is designed to support the storage of multiple biometric identifier records for given user to provide increased system accuracy.

The HA-API is broken into three sections: Biometric Technology functions, Biometric Authentication functions, and Biometric Utility functions. The Biometric Technology functions provide the ability to specify Biometric Technologies. The Biometric Technology is later used in processing and verifying biometric data. The Biometric Authentication functions provide the ability to capture, process and match Biometric Data retrieved from different devices and using different technologies. The Biometric Utility functions provide various tools to facilitate programming, including access to vendor specific options.

4.1 Biometric Technology Functions

4.1.1 EnumBioTechnology

HAAPIERROR EnumBioTechnology(LPBIOTECH pBioTechEnum,
DWORD dwBuf,
LPDWORD pdwNeeded,
LPDWORD pdwReturned)

Description:

This function enumerates the biometric technologies installed on a system. It identifies which specific biometric technologies are available for use by the application.

Parameters:

pBioTechEnum – A pointer to an array of Biometric Technology structures. For more information on Biometric Technology structures, refer to ***Section 6.0, Structures***. This parameter can be NULL.

dwBuf – Specifies the number of elements in the array pointed to by pBioTechEnum.

pdwNeeded – Pointer to a value that receives the number of Biometric Technology structures copied if the function succeeds or the number of strucutures required if dwBuf is too small.

pdwReturned – Pointer to a value that receives the number of Biometric Technology structures that the function returns in the array to which pBioTechEnum points.

Return Value:

Upon success, this function returns a HAAPI_NOERROR. If the function fails, it returns a HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

If pBioTechEnum is NULL, the function will return the number of Biometric Technologies installed on the PC in the parameter pdwNeeded. This value can be used to allocate the size of the array pointed to by pBioTechEnum which can then be used in a subsequent call to ***EnumBioTechnology***.

4.1.2 GetBioTechnology

HBT GetBioTechnology(BUID buidBioTechnology)

Description:

This function is used to initialize a biometric technology. The resulting handle to the biometric technology can later be used by the Biometric Authentication functions, ***HAAPIProcess*** and ***HAAPIVerify***.

Parameters:

buidBioTechnology – A unique biometric identifier for a specific technology. A list of available BUID's and their description can be retrieved using ***EnumBioTechnology***.

Return Value:

Upon successful initialization of a Biometric Technology, ***GetBioTechnology*** returns a Biometric Technology Handle. If this function fails, this function HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

This Biometric Technology Handle must be released using ***ReleaseBioTechnology***.

4.1.3 ReleaseBioTechnology

HAAPIERROR ReleaseBioTechnology(HBT hbtBioTechnology)

Description:

This function frees a handle created by ***GetBioTechnology***. Once released, to use the biometric technology again requires that the ***GetBioTechnology*** be invoked.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology.

Return Value:

Upon success, this function returns a HAAPI_NOERROR. If the function fails, it returns a HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

4.2 Biometric Authentication Functions

4.2.1 HAAPICapture

HAAPIERROR HAAPICapture(HBT hbtBioTechnology,
LPSCREENATTRIBS lpScreenAttribs,
LPRAWBIODATA lpRawBioData)

Description:

This function captures Raw Biometric Data, specified by the Biometric Technology passed as parameters. The application is presented with a modal capture dialog box that varies according to the technology being used. Upon successful capture, a Raw Biometric Data structure is filled with appropriate data. Depending on the technology, this data has unique characteristics. It can, in speaker verification for instance, be a series of speech utterances.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to null.

lpRawBioData – a pointer to a Raw Biometric Data structure that is filled upon successful capture. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***HAAPICapture*** returns HAAPI_NOERROR. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

This function allocates the raw data pointed to by the lpRawData member of the Raw Biometric Data structure. It is the application programmers responsibility to free this allocated memory using the function ***HAAPIFree***.

The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data structure.

4.2.2 HAAPIProcess

HAAPIERROR HAAPIProcess(HBT hbtBioTechnology,
LPRAWBIO lpRawBioData,
LPBIR lpBioIDRec)

Description:

This function processes the raw biometric data captured via a call to ***HAAPICapture*** and extracts a unique Biometric Identifier Record. This Raw Biometric Data contains raw data and data size. The raw data differs per technology, for example: Finger Imaging could be – a raw grayscale finger image; Facial Recognition could be – a video image of a face; Speaker Verification could be – a digitized speech waveform. The resulting Biometric Identifier Record contains processed data and data size. The processed data also differs per technology, for example: Finger Imaging could be – a finger image identifier record; Facial Recognition could be – a facial image or feature data; Speaker Verification could be – a spectral representation.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpRawBioData – a pointer to a Raw Biometric Data structure. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpBioIDRec – a pointer to a Biometric Identifier Record. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful extraction of a Biometric Identifier Record, ***HAAPIProcess*** returns HAAPI_NOERROR. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

This function allocates the processed data pointed to by the lpBioData member of the Biometric Identifier Record structure. It is the application programmers responsibility to free this allocated memory using the function ***HAAPIFree***.

The size of the data allocated is indicated by the ulSize member of the Biometric Identifier Record structure.

4.2.3 HAAPIVerify

HAAPIERROR HAAPIVerify(HBT hbtBioTechnology,
LPBIR lpSampleBIR,
LPBIR lpStoredBIR,
LPBIR lpAdaptedBIR,
LPBOOL lpbResponse)

Description:

This function performs a verification (1-to-1) match against two Biometric Identifier Records. The first BIR is the sample biometric captured at time of verification. The second stored BIR is retrieved from a

database for verification. If the stored BIR is modified as a result of the verification, the modified or adapted BIR is returned.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpSampleBIR – a pointer to the Biometric Identifier Record structure in question. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpStoredBIR – a pointer to the original Biometric Identifier Record structure stored at enrollment. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpAdaptedBIR – a pointer to the an adapted Biometric Identifier Record structure, based upon the original BIR stored at enrollment and the sample BIR taken for verification. This parameter can be NULL if an adapted BIR is not desired. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpbResponse – a pointer to a Boolean value indicating (TRUE/FALSE) indicating whether the BIRs matched or not.

Return Value:

Upon successful execution, ***HAAPIVerify*** returns HAAPI_NOERROR. To determine if the BIRs matched, examine the parameter, lpbResponse. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

If an adapted BIR is desired, check ***GetLastError*** to determine if an adaptation has been performed. If so, it will be set to HAAPI_ADAPTEDBIR. If set, check lpAdaptedBIR for the location of the adapted BIR data.

In the event that adaptation is not supported for a given biometric, HAAPI_ADAPTATIONNOTSUPPORTED is returned by ***GetLastError***.

Remarks:

If a Biometric Identifier Record is passed in as the parameter lpStoredBIR, and the match is successful, ***HAAPIVerify*** may attempt to adapt the enrolled BIR with information taken from the sample BIR. (Some BSPs may adapt and others may not). The resulting lpAdaptedBIR should now be considered an optimal enrollment, and be saved to the enrollment database. (It is up to the application whether or not it uses or discards this data). It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function allocates the data pointed to by the lpBioData member of the Biometric Identification Data structure. It is the application programmer's responsibility to free this allocated memory using the function ***HAAPIFree***.

The ulSize member of the Biometric Identification Record structure indicates the size of the data allocated.

The Data stored in the Biometric Identification Record structure can be a concatenation of multiple identifiers.

Setting of verification thresholds or returning of verification matching scores is provided using the ***HAAPIInformation*** function.

4.2.4 HAAPILiveVerify

HAAPIERROR HAAPILiveVerify(HBT hbtBioTechnology,
LPSCREENATTRIBS lpScreenAttribs,
LPBIRlpStoredBIR,
LPBIR lpAdaptedBIR,
DWORD dTimeout,
LPBOOL lpbResponse)

Description:

This function encapsulates the functionality of ***HAAPICapture***, ***HAAPIProcess***, and ***HAAPIVerify***. It is intended to be used for technologies that perform continuous capture, process, and verification until a match is found or a timeout is reached. ***HAAPILiveVerify*** can also be used as a convenience function, tying the three functions commonly called in succession.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to null.

lpStoredBIR– a pointer to the original Biometric Identifier Record structure stored at enrollment. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpAdaptedBIR – a pointer to the an adapted Biometric Identifier Record structure, based upon the original BIR stored at enrollment and the sample BIR taken for verification. This parameter can be NULL if an adapted BIR is not desired. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

dTimeout – a double specifying the timeout value (in milliseconds) for the Live Verify. If this timeout is reached before a valid biometric is captured and verified, the function returns an error. This value can be any positive number or HAAPI_NOTIMEOUT.

lpbResponse – a pointer to a Boolean value indicating (TRUE/FALSE) indicating whether the BIRs matched or not.

Return Value:

Upon successful capture and match, ***HAAPILiveVerify*** returns HAAPI_NOERROR and sets the parameter lpbResponse to TRUE. If this function fails or does not successfully match in the time specified, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

If an adapted BIR is desired, check ***GetLastError*** to determine if an adaptation has been performed. If so, it will be set to HAAPI_ADAPTEDBIR. If set, check lpAdaptedBIR for the location of the adapted BIR data.

In the event that adaptation is not supported for a given biometric, HAAPI_ADAPTATIONNOTSUPPORTED is returned by ***GetLastError***.

Remarks:

If a Biometric Identifier Record is passed in as the parameter lpStoredBIR, and the match is successful, ***HAAPILiveVerify*** may attempt to adapt the enrolled BIR with information taken from the sample BIR. (Some BSPs may adapt and others may not). The resulting lpAdaptedBIR should now be considered an optimal enrollment, and be saved to the enrollment database. (It is up to the application whether or not it uses or discards this data). It is important to note that adaptation may not occur in all cases.

In the event of an adaptation, this function allocates the data pointed to by the lpBioData member of the Biometric Identification Data structure. It is the application programmer's responsibility to free this allocated memory using the function ***HAAPIFree***.

The ulSize member of the Biometric Identification Record structure indicates the size of the data allocated.

The Data stored in the Biometric Identification Record structure can be a concatenation of multiple identifiers.

Setting of verification thresholds or returning of verification matching scores is provided using the ***HAAPIInformation*** function.

4.2.5 HAAPIEnroll

```
HAAPIERROR HAAPIEnroll(HBT hbtBioTechnology,
                       DWORD dwEnrollType,
                       LPSCREENATTRIBS lpScreenAttribs,
                       LPRAWBIODATA lpRawBioData,
                       LPBIR lpBioIDRec,
                       LPENROLLMENTPAGES lpPages)
```

Description:

This function captures and processes Raw Biometric Data, specified by the Biometric Technology passed as parameters. This function differs from ***HAAPICapture*** because it encapsulates the entire process of enrollment (i.e., capture, process). Batch enrollment is also supported. A wizard provides the means for the application to ensure a successful

enrollment. The application is presented with a modal wizard dialog box, that varies according to the technology being used, that guides the user through the various steps of enrollment.

The ***HAAPIUpdate*** function (see Section 4.2.6) is used to update a previously enrolled user by overwriting, appending, or averaging the old with the new data (the specific update scheme being technology dependent).

The wizard provided by the ***HAAPEnroll*** function is extendable. Through the use of the Win32 Common Control Property Page and the PSH_WIZARD flag, the wizard can easily have pages added to it. The parameter lpPages provides the structure for the developer to add pages.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

dwEnrollType – a value which indicates what data is to be returned (i.e., what type of enrollment is to be performed) and is a composite set of subflags having the following meaning:

Value	Meaning
ENROLL_CAPTURE	If included, indicates that a capture will be performed.
ENROLL_BIR	If included, indicates that a BIR will be returned.
ENROLL_RAW	If included, indicates that raw data will be returned.

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to null.

lpRawBioData – a pointer to a Raw Biometric Data structure that is filled upon successful capture. If this parameter is NULL, the Raw Biometric Data is discarded upon function return. If the pointer is valid, but the lpRawData member of this structure is NULL, then the Raw Biometric

Data from the enrollment is returned. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpBioIDRec – a pointer to a Biometric Identification Data structure that is filled upon successful capture/process. This parameter can be NULL. For more information on Biometric Identification Data structures, see ***Section 6.0, Structures***.

lpPages – a pointer to an Enrollment Pages Data structure. This parameter defines any additional pages that you would like to add to the Enrollment Wizard. Pages can be added at the beginning or at the end of the Wizard. Wizard pages are Win32 Common Control Property Pages with the PSH_WIZARD flag set. For more information about Property Pages and the PSH_WIZARD flag, please refer to the Microsoft Platform SDK, Wizard Property Sheets. For more information on Enrollment Pages Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpPages will be set to null.

Return Value:

Upon successful capture, ***HAAPIEnroll*** returns HAAPI_NOERROR. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

Storage of returned enrollment data (i.e. database, smart card, etc) is the responsibility of the application program.

This function allocates the raw data pointed to by the lpRawData member of the Raw Biometric Data (lpRawBioData) structure and by the lpBioData member of the Biometric Identification data (lpBioIDRec) structure. It is the application programmers responsibility to free this allocated memory using the function ***HAAPIFree***. The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data and Biometric Identification Record structures.

HAAPIEnroll can function in several different fashions, depending on the values included in the dwEnrollType parameter. Enrollment types fall into the following four categories:

Standard Enrollment #1 - This is the most commonly used enrollment type. It involves capturing the raw biometric data, processing it, and returning the processed data in the form of a BIR. For the standard enrollment #1, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_BIR.

Standard Enrollment #2 - This is the same as standard enrollment #1, but this returns the newly captured raw biometric data as well as the BIR. For the standard enrollment #2, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_RAW | ENROLL_BIR.

Batch Enrollment, Part 1 - This allows raw biometric data to be captured for later processing and storage. It returns raw biometric data. For the batch enrollment part 1, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_RAW.

Batch Enrollment, Part 2 - This performs the processing of previously captured raw biometric. It returns a processed BIR. For the batch enrollment part 2, dwEnrollType would be:

ENROLL_BIR.

To perform enrollment processing only (Batch Enroll, Part 2), the application must pass in the previously captured raw biometric data via lpRawBioData. This enrollment type typically requires no GUI.

NOTE: The data stored in the Biometric Identification Record structure (lpBioIDRec) is technology dependent. See the BSP documentation for the details of what is included in this structure.

4.2.6 HAAPIUpdate

HAAPIERROR HAAPIUpdate(HBT hbtBioTechnology,
LPRAWBIODATA lpOldRawBioData,
LPRAWBIODATA lpNewRawBioData,
LPBIR lpStoredBioIDRec,
LPBIR lpNewBioIDRec)

Description:

This function performs various adaptation, reenrollment or updates to previously enrolled/stored biometric data. All data collection must be done previously through ***HAAPICapture*** or ***HAAPIEnroll***. This function provides the capability to receive raw biometric data. In addition, a reenrollment may require the current stored BIR in order to create the new one. Thus, this function extends the ***HAAPIProcess*** functionality. The ***HAAPIUpdate*** may replace, append, or average the old with the new data (the specific update scheme being technology dependent).

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpOldRawBioData – a pointer to a Raw Biometric Data structure. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpNewRawBioData – a pointer to a recently captured Raw Biometric Data structure either from a ***HAAPIEnroll*** or a ***HAAPICapture***. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpStoredBioIDRec – a pointer to a Biometric Identification Data structure that has the previously stored enrollment that will be adapted by this call to ***HAAPIUpdate***. For more information on Biometric Identifier Data structures, see ***Section 6.0, Structures***.

lpNewBioIDRec – a pointer to a Biometric Identification Data structure that is filled with the new updated/ refreshed/ adapted BIR For more information on Biometric Identifier Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***HAAPIUpdate*** returns HAAPI_NOERROR. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data and Biometric Identifier Record structures.

Note that as a minimum, the application must pass in the newly captured raw biometric data (lpNewRawBioData) and the BSP must pass back the new biometric identifier record (lpNewBioIDRec). The other data (lpOldRawBioData and lpStoredBioIDRec) is set to null if unused. Use is dependent on the biometric technology (see BSP documentation for details).

4.2.7 HAAPIIdentify

This is a place-holder for a 1:N search/match function to be potentially added into future releases of this specification.

4.3 Biometric Utility Functions

4.3.1 HAAPIFree

HAAPIERROR HAAPIFree(HBT hbtBioTechnology,
LPVOID lpData)

Description:

This function releases memory that has been allocated through any of the HAAPI Authentication functions

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpData – a pointer to the data member which was allocated through one of the authentication functions.

Return Value:

Upon successful capture, ***HAAPIFree*** returns HAAPI_NOERROR. If this function fails, this function returns

HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

This function MUST be used to free the memory allocated by any of the Authentication functions. DO NOT pass in the full BIR or RAWBIODATA structure. This function only expects the data member of the structure.

4.3.2 HAAPIInformation

HAAPIERROR HAAPIInformation(HBT hbtBioTechnology,
long lVendorInfo,
LPVOID lpData,
DWORD cbSize)

Description:

This function provides an interface for the application developer to get and set information specific to a Biometric Technology. It must also be noted that vendors are only required to support a minimal set of constants for the requested information parameter. Any parameters used outside of this set are considered specific to the vendor and may not work across multiple vendor applications.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lVendorInfo – a long integer constant defining the information being retrieved or set.

lpData – a pointer to the variable that contains the data to be set or will hold the data being retrieved.

cbSize – the size, in bytes, of lpData.

Return Value:

Upon successful capture, ***HAAPIInformation*** returns HAAPI_NOERROR. If this function fails, this function returns

HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

All vendors must support the following constants:

Constant	Data Type for lpData
HAAPI_MAJORVERSION	long (read only)
HAAPI_MINORVERSION	long (read only)
HAAPI_BUILDVERSION	long (read only)
HAAPI_BUILDDATE	long (read only – julian date)
HAAPI_VENDORNAME	LPTSTR (read only)
HAAPI_TECHNAME	LPTSTR (read only)

Variable	Data Type for lpData
HAAPI_GET_LAST_SCORE	Scoring Record (read only)
HAAPI_SET_SYS_THRESHOLD	Threshold Record (write)
HAAPI_GET_SYS_THRESHOLD	Threshold Record (read)

Notes: 1) 'read only' is from the perspective of the application.
2) Variable definitions are found in Section 6.0, Structures.

Altering information using this function can adversely affect the performance and matching of a particular Biometric Technology. Care should be exercised when using this function. See Appendix F for a discussion on scoring and thresholding.

For example, whether or not two BIRs are determined to match or not is dependent on the threshold setting. It is important to realize that if the threshold value is changed, there could be a dramatic change in the amount of false accepts or false rejects experienced. The vendor's pre-set thresholds should be adequate in most instances.

Vendors may provide additional constants/variables that can be read/write. It is important to remember that these additional constants will be specific to that vendor.

This function allocates data pointed to by lpData. It is the application programmers responsibility to free this allocated memory using the function ***HAAPIFree*** for readable data

Under Windows NT, any strings returned by this function are Wide Character (UNICODE) strings. If UNICODE is not defined in your

project, use the Win32 API WideCharToMultiByte to convert the string to ANSI Code Page.

Under Windows 95, any strings returned by this function are ANSI strings.

4.3.3 HAAPIBioProperties

HAAPIERROR HAAPIBioProperties(HBT hbtBioTechnology,
LPSCREENATTRIBS lpScreenAttribs)

Description:

This function displays the biometric properties dialog box, if it is available. The biometric properties dialog box could be used to read and set parameters, which are specific to a biometric and their input devices.

Parameters:

hbtBioTechnology – a handle to a Biometric Technology. This handle is returned by a call to ***GetBioTechnology***.

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***HAAPIBioProperties*** returns HAAPI_NOERROR. If this function fails, this function returns HAAPI_ERROR. To get extended error information, call the Win32 API, ***GetLastError***.

Remarks:

This function is intended to provide access to unique vendor technology properties and device settings. An example would include adjustment of contrast or brightness for an imaging device.

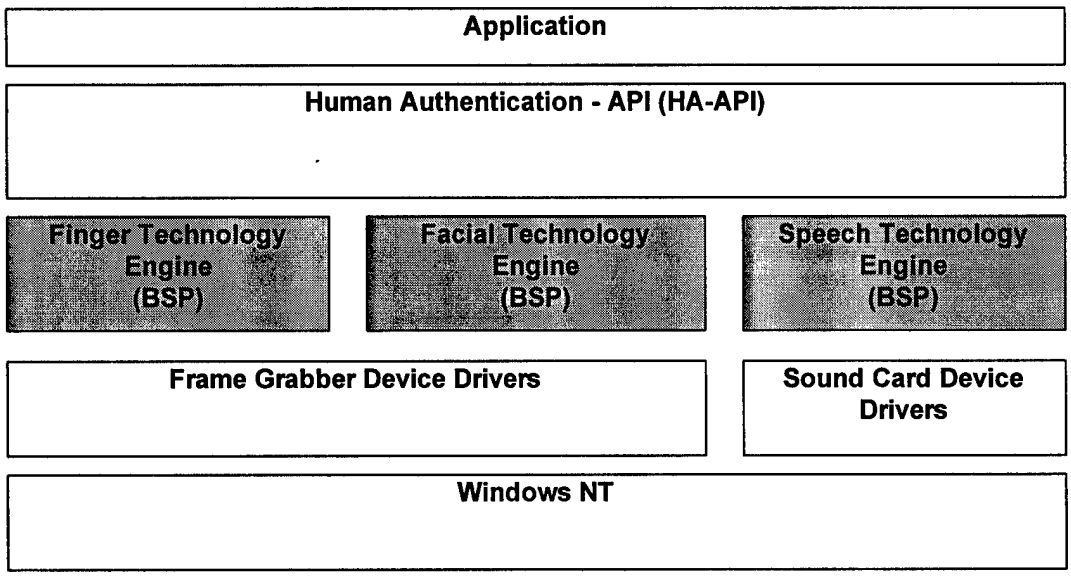
In the event that a biometric properties page is not supported for a given biometric, HAAPI_NOBIOPROPERTIESPAGE is returned by ***GetLastError***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to null.

5.0 Service Provider Interface

5.1 Description

While the HA-API provides a generic biometric interface for application developers to use, each specific biometric vendor is responsible for providing certain specific key elements for the API, encompassed in a Technology Engine. As an example, the diagram below describes a possible HA-API application architecture (the biometric technology engines are highlighted):



Encompassed in each Technology Engine, the vendor must provide exported functions that perform the following:

1. An enrollment wizard.
2. A standard biometric capture screen.
3. An interface for processing the identifier data from captured biometric data.
4. An interface for verifying two sets of identifier data.
5. An interface performing a Live Verify (capture, process and verify).
6. An interface to retrieve and set module specific information.

This technology must be entered in the registry to make the HA-API aware of its existence. To register a technology, a vendor must first generate a unique biometric

identifier (BUID). The BUID is a 128 bit Global Unique Identifier (GUID). This value, along with the technology module (BSP) name and the technology name, must be stored in the system registry. The registry entries are:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\BUID**
**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\ModuleName**
**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
*TechnologyName***

The BUID, BSP Module Name and the Technology Name are string values. Please refer to ***Section 5.1.1, BUID,*** for an example of how a BUID should be stored in the registry.

Along with these entries must exist the names of the enrollment function, the update function, the capture function, the processing function, the verifying function, the live verify function, the information function, the biometric properties function, and the free function. These entries are described in the next sections. All device initialization necessary for the particular vendor/technology must occur within these functions.

5.1.1 BUID

The BUID is a unique identifier that is generated for every technology module (BSP) that is being released. It is used to identify a specific BSP product and allows an application to distinguish among multiple BSPs. This identifier is a vendor generated GUID. Globally Unique Identifiers or GUIDs are 128 bit numbers and when generated properly, never produce the same number twice, no matter how many times it is run or how many different machines it runs on. Every entity that needs to be uniquely identified (such as each BSP and/or major revision thereof) should have a BUID. For example, a particular vendor could generated a GUID of:

{4E43227D-A799-11d1-AFAF-00609761BD69}

and therefore use this as the HAAPI Biometric Unique Identifier for their module. This identifier as shown above, with enclosed in braces, is stored in the registry as a string value.

5.1.2 Asynchronous Operation

All BSP vendor module software must support asynchronous operation in that they must be thread-safe.

5.1.3 Authentication of BSP module

Similar to Cryptographic Service Providers (CSPs), the biometric community should work toward requiring every BSP be digitally signed by Microsoft in order to be recognized by the operating system. For CSPs, the operating system validates the signature periodically to ensure that the CSP has not been tampered, removed or replaced. For the non-Microsoft operating system environments, other methods exist to jointly authenticate the CSP to some overall system security module. The analogy may be a cryptographic solution between the HA-API runtime and the BSP for authenticating each other.

5.1.4 ANSI vs. UNICODE

To facilitate HAAPI availability under both Windows 95 and Windows NT, the BSP vendor should support both ANSI and UNICODE strings. Under Windows 95, any function that returns a character string, must return it as a ANSI string. Under Windows NT, these strings must be returned as UNICODE. This ensures persistence of string data across all vendor BSPs.

5.2 User Interface

To maintain a high level of GUI consistency throughout HA-API product implementations, all user interface screens, including the enrollment wizard, should have a standard look and feel. The vendor must adhere to the Windows Interface Guidelines for Software Design for layout, color, fonts, input/output mechanisms, etc. Samples of compliant are provided in Appendix G as a guide. Additionally, the following guidelines are provided:

a. Keep all screens and windows as simple as possible
b. Windows should perform a single function
c. Keep nesting to the minimum practical
d. Use the wizard technique for functions composed of a series of steps, using the Common Control Wizard Property Pages.
e. Default selections whenever possible

It is recognized that not all biometric applications are implemented on PC workstations and thus provide a GUI. For such embedded applications (e.g., door opening devices,

voice over telephone, etc.), these guidelines would obviously not apply. However, for most computer applications, and whenever a GUI is used, the above guidelines apply.

5.3 The Enrollment Wizard

The enrollment wizard provides the ability to perform biometric specific enrollments. This wizard is initiated by a call to the HA-API function, ***HAAPIEnroll***. The wizard should be able to perform any number of biometric captures. After these captures have occurred, it should also perform the necessary identifier extraction. Both sets of data (the raw biometric data and the biometric identifier record) should be returned. To accommodate a standard interface for the users, two generic structures have been created: Raw Biometric Data structure and Biometric Identifier Record structure. These structures both have two members, a size member and a data member. The data member points to a block of memory that contains all of the biometric data. (If there are multiple images, they should be stored together in the same block of memory) It is also suggested that this data should be prefixed by a vendor specific structure, containing biometric specific data (i.e., number of images, image size, camera type, frame grabber type, etc.). To accommodate easy memory management, this pointer should be able to be freed by the programmer via a call to ***FreeFn.*** The size member should tell the programmer the exact size in bytes of the data member. For a diagram of the Raw Biometric Data Structure and the Biometric Identifier Record structure, see ***Section 6.0, Structures***.

User interface guidelines applicable to the enrollment wizard are provided in Section 5.2 above. It is imperative that the vendor use the Win32 Common Control Property Page wizard sheets to implement the Enrollment wizard. Not only does this make the development effort easier, it provides a cohesive method of developing a wizard which is standard across all HAAPI vendor modules.

A Vendor **must** provide an exported enrollment function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\EnrollmentFunctionName**

5.3.1 Exported Enrollment Function

```
HAAPIERROR EnrollmentFn(DWORD dwEnrollType,
                        LPSCREENATTRIBS lpScreenAttribs,
                        LPRAWBIODATA lpRawBioData,
                        LPBIR lpBioIDRec,
                        LPENROLLMENTPAGES lpPages)
```

Description:

This function captures and processes Raw Biometric Data. This function differs from ***CaptureFn*** (see 5.5.1, below) because it encapsulates the entire process of enrollment. A wizard provides the means for the application to ensure a successful enrollment. The application is presented with a modal wizard dialog box, that varies according to the technology being used, that guides the user through the various steps of enrollment. ***UpdateFn*** is used to update a previously enrolled user.

Batch enrollment can be accommodated by this function. ***EnrollmentFn*** optionally allows the return of Raw Biometric Data (without processing) and/or to conversely pass in Raw Biometric Data for processing without any user interface.

The wizard provided by the ***EnrollmentFn*** function must be extendable, through the use of the Win32 Common Control Property Page and the PSH_WIZARD flag via the lpPages structure.

Parameters:

EnrollType – a value which indicates what data is to be returned (i.e., what type of enrollment is to be performed) and is composed of an 'OR'ing of one or more of the following values as follows:

Value	Meaning
ENROLL_CAPTURE	If set, indicates that a capture will be performed.
ENROLL_BIR	If set, indicates that a BIR will be returned.
ENROLL_RAW	If set, indicates that raw data will be returned.

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to NULL.

lpRawBioData – a pointer to a Raw Biometric Data structure that is filled upon successful capture. If this parameter is NULL, the Raw

Biometric Data is discarded upon function return. If the pointer is valid, but the lpRawData member of this structure is NULL, then the Raw Biometric Data from the enrollment is returned. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpBioIDRec – a pointer to a Biometric Identification Data structure that is filled upon successful capture/process. This parameter can be NULL. For more information on Biometric Identification Data structures, see ***Section 6.0, Structures***.

lpPages – a pointer to an Enrollment Pages Data structure. This parameter defines any additional pages that you would like to add to the Enrollment Wizard. Pages can be added at the beginning or at the end of the Wizard. Wizard pages are Win32 Common Control Property Pages with the PSH_WIZARD flag set. For more information about Property Pages and the PSH_WIZARD flag, please refer to the Microsoft Platform SDK, Wizard Property Sheets. For more information on Enrollment Pages Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***EnrollmentFn*** returns HAAPI_NOERROR. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***.

Remarks:

This function allocates the raw data pointed to by the lpRawData member of the Raw Biometric Data structure and by the lpBioData member of the Biometric Identification data (lpBioIDRec) structure. It is the application programmers responsibility to free this allocated memory using the ***FreeFn*** function.

The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data and Biometric Identification Record structures.

EnrollmentFn can function in several different fashions, depending on the values included in the EnrollType parameter. Enrollment types fall into the following four categories:

Standard Enrollment #1 - This is the most commonly used enrollment type. It involves capturing the raw biometric data, processing it, and returning the processed data in the form of a BIR. For the standard enrollment #1, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_BIR.

Standard Enrollment #2 - This is the same as standard enrollment #1, but this returns the newly captured raw biometric data as well as the BIR. For the standard enrollment #2, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_RAW | ENROLL_BIR.

Batch Enrollment, Part 1 - This allows raw biometric data to be captured for later processing and storage. It returns raw biometric data. For the batch enrollment part 1, dwEnrollType would be:

ENROLL_CAPTURE | ENROLL_RAW.

Batch Enrollment, Part 2 - This performs the processing of previously captured raw biometric. It returns a processed BIR. For the batch enrollment part 2, dwEnrollType would be:

ENROLL_BIR.

To perform enrollment processing only (Batch Enroll, Part 2), the application must pass in the previously captured raw biometric data via lpRawBioData. This enrollment type typically requires no GUI.

NOTE: The data stored in the Biometric Identification Record structure (lpBioIDRec) is technology dependent. BSP documentation must include details of what is included in this structure.

The Data stored in the Biometric Identification Record structure (lpNewBioIDRec and lpCurrentBioIDRec) can be a concatenation of multiple identifiers, and can contain both raw and processed data, as well as other information. The following is suggested:

- The vendor should return for storage any data that will be required for subsequent matching purposes.
- The vendor should include necessary header data to allow discernment as to the content of the structure. For example, items that may be useful include:
    - Structure type (lpRawData, lpBioData)
    - Data type (raw, processed, both)
    - Number of captures
    - BUID (of enrolling technology)
    - Capture date/time

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs and lpPages will be set to NULL.

5.4 Update Enrollment

This function provides for the update of previously enrolled biometric data. The previous data may be replaced, appended, or in some way averaged with the newly captured data to result in a new biometric identification record. Use of this function will generally be performed immediately after a capture or enrollment, using this newly captured data along with the previous data to calculate the new biometric identifier.

A Vendor **must** provide an exported update function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\UpdateFunctionName**

5.4.1 Exported Update Function

HAAPIERROR UpdateFn(LPRAWBIODATA lpOldRawBioData,
LPRAWBIODATA lpNewRawBioData,
LPBIR lpStoredBioIDRec,
LPBIR lpNewBioIDRec)

Description:

This function performs various adaptation, reenrollment or updates to previously enrolled/stored biometric data. All data collection must be done previously through ***HAAPICapture*** or ***HAAPIEnroll***. This function provides the capability to receive raw biometric data. In addition, a reenrollment may require the current stored BIR in order to create the new one. Thus, this function extends the ***HAAPIProcess*** functionality. The ***HAAPIUpdate*** may replace, append, or average the old with the new data (the specific update scheme being technology dependent).

Parameters:

lpOldRawBioData – a pointer to a Raw Biometric Data structure. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpNewRawBioData – a pointer to a recently captured Raw Biometric Data structure. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpStoredBioIDRec – a pointer to a Biometric Identification Data structure that has the previously stored enrollment that will be adapted by this call to ***UpdateFn***. For more information on Biometric Identifier Data structures, see ***Section 6.0, Structures***.

lpNewBioIDRec – a pointer to a Biometric Identification Data structure that is filled with the new updated/ refreshed/ adapted BIR For more information on Biometric Identifier Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***UpdateFn*** returns HAAPI_NOERROR. . If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***.

Remarks:

The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data and Biometric Identifier Record structures.

Note that as a minimum, the application must pass in the newly captured raw biometric data (lpNewRawBioData) and the BSP must pass back the new biometric identifier record (lpNewBioIDRec). The other data (lpOldRawBioData and lpStoredBioIDRec) is set to NULL if unused. Use is dependent on the biometric technology. The BSP documentation must identify which of the optional data is used, if any.

5.5 Standard Capture Screen

The standard biometric capture screen should challenge the user to capture the biometric. It simply returns a Biometric Identifier Record structure that can be used to extract and match against.

User interface guidelines applicable to the enrollment wizard are provided in Section 5.2 above.

A Vendor **must** provide an exported capture function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\CaptureFunctionName**

5.5.1 Exported Capture Function

HAAPIERROR CaptureFn(LPSCREENATTRIBS lpScreenAttribs,
LPRAWBIODATA lpRawBioData)

Description:

This function captures Raw Biometric Data. The application is presented with a modal capture dialog box, that varies according to the technology being used. Upon successful capture, a Raw Biometric Data structure is filled with appropriate data. Depending on the technology, this data has unique characteristics. It can, in Speaker Verification for instance, be a series of speech utterances.

Parameters:

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to NULL.

lpRawBioData – a pointer to a Raw Biometric Data structure that is filled upon successful capture. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful capture, ***CaptureFn*** returns HAAPI_NOERROR. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***.

Remarks:

This function allocates the raw data pointed to by the lpRawData member of the Raw Biometric Data structure. The application programmer will free this allocated memory through the ***FreeFn*** function.

The size of the data allocated is indicated by the ulSize member of the Raw Biometric Data structure.

5.6 Processing Interface

The processing interface should accept a Raw Biometric Data structure and return a Biometric Identifier Record structure.

A Vendor **must** provide an exported processing function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\ProcessFunctionName**

5.6.1 Exported Process Function

HAAPIERROR ProcessFn(LPRAWBIO lpRawBioData,
LPBIR lpBioIDRec)

Description:

This function processes the raw biometric data captured via a call to ***HAAPICapture*** and extracts a unique Biometric Identifier Record. The Raw Biometric Data contains raw data and data size. The raw data differs per technology, for example: Finger Imaging could be – a raw grayscale finger image; Facial Recognition could be – a video image of a face; Speaker Verification could be – a digitized speech waveform. The resulting Biometric Identifier Record contains processed data and data size. The processed data also differs per technology, for example: Finger Imaging could be – a finger image identifier record; Facial Recognition could be – facial image data; Speaker Verification could be – a spectral representation.

Parameters:

lpRawBioData – a pointer to a Raw Biometric Data structure. For more information on Raw Biometric Data structures, see ***Section 6.0, Structures***.

lpBioIDRec – a pointer to a Biometric Identifier Record. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

Return Value:

Upon successful extraction of a Biometric Identifier Record, ***ProcessFn*** returns HAAPI_NOERROR. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***.

Remarks:

This function allocates the raw data pointed to by the lpBioData member of the Biometric Identifier Record structure. The application programmer will free this allocated memory through the ***FreeFn*** function.

The size of the data allocated is indicated by the ulSize member of the Biometric Identifier Record structure.

5.7 Verify Interface

The verify interface should accept two Biometric Identifier Record structures. The interface should return a Yes/No answer for the match.

A Vendor **must** provide an exported verify function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\VerifyFunctionName**

5.7.1 Exported Verify Function

```
HAAPIERROR VerifyFn(LPBIR lpSampleBIR,
                    LPBIR lpStoredBIR,
                    LPBIR lpAdaptedBIR,
                    LPBOOL lpbResponse)
```

Description:

This function performs a verification (1-to-1) match against two Biometric Identifier Records. The first BIR is the sample biometric captured at time of verification. The second stored BIR is retrieved from a database for verification. If the stored BIR is modified as a result the verification, the modified or adapted BIR is returned.

Parameters:

lpSampleBIR – a pointer to the Biometric Identifier Record structure in question. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpStoredBIR – a pointer to the original Biometric Identifier Record structure stored at enrollment. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpAdaptedBIR – a pointer to the an adapted Biometric Identifier Record structure, based upon the original BIR stored at enrollment and the sample BIR taken for verification. This parameter can be NULL if the application does not want an adapted BIR returned. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpbResponse – a pointer to a Boolean value indicating (TRUE/FALSE) indicating whether the BIRs matched or not.

Return Value:

Upon successful execution, ***VerifyFn*** returns HAAPI_NOERROR. If the BIRs match, set the parameter lpbResponse to TRUE. If the BIR's do not match, set it to FALSE. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***. If adaptation is performed, place the new BIR into LPAdaptedBIR and ***SetLastError*** to HAAPI-ADAPTEDBIR. In the event that adaptation is not supported, ***SetLastError*** to HAAPI_ADAPTATIONNOTSUPPORTED.

Remarks:

If a Biometric Identifier Record is passed in as the parameter lpStoredBIR, and the match is successful, ***VerifyFn*** may attempt to adapt the enrolled BIR with information taken from the sample BIR. (It is up to the BSP vendor whether or not to provide adaptation.) If an adaptation is performed, the lpBioData member of the Biometric Identification Data structure must point to a valid address upon return from the function.

Also, upon a successful match, set the extended error to HAAPI_ADAPTEDBIR.

In the event of an adaptation, this function allocates the data pointed to by the lpBioData member of the Biometric Identification Data structure. It is the application programmer's responsibility to free this allocated memory using the ***FreeFn*** function.

The ulSize member of the Biometric Identification Record structure indicates the size of the data allocated.

The Data stored in the Biometric Identification Record structure can be a concatenation of multiple identifiers.

Setting of verification thresholds or returning of verification matching scores may be provided by individual biometric technology vendors. If so, access is provided using the exported ***InformationFn*** function.

5.8 LiveVerify Interface

The live verify interface should accept a single, stored Biometric Identifier Record structure. The interface should return a Yes/No answer for the match.

A Vendor **must** provide an exported live verify function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\LiveVerifyFunctionName**

5.8.1 Exported Live Verify Function

```
HAAPIERROR LiveVerifyFn(LPSCREENATTRIBS lpScreenAttribs,
                        LPBIRlpStoredBIR,
                        LPBIR lpAdaptedBIR,
                        int iTimeout,
                        LPBOOL lpbResponse)
```

Description:

This function performs a live verification (1-to-1). A live verification consists of capturing, processing, and matching against a stored Biometric Identifier Record. There are two reasons for a live verification function. It can be used as a convenience function, combining three commonly used, successively called functions. ***LiveVerifyFn*** can also be used to perform repeated captures, processes and verifies, until a match is successful or a timeout is reached.

Parameters:

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to NULL.

lpStoredBIR – a pointer to the original Biometric Identifier Record structure stored at enrollment. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

lpAdaptedBIR – a pointer to the an adapted Biometric Identifier Record structure, based upon the original BIR stored at enrollment and the sample BIR taken for verification. This parameter can be NULL if the application does not want an adapted BIR returned. For more information on Biometric Identifier Record structures, see ***Section 6.0, Structures***.

iTimeout – an integer designating the timeout (in milliseconds) to be used for the live verify. This can be a positive value or HAAPI_NOTIMEOUT.

lpbResponse – a pointer to a Boolean value indicating (TRUE/FALSE) indicating whether the BIRs matched or not.

Return Value:

Upon successful capture and match, ***LiveVerifyFn*** returns HAAPI_NOERROR and sets lpbResponse to TRUE. If the function fails, either through timeout, cancel, or no match, return the appropriate HAAPI_ERROR via a call to the Win32 API, ***SetLastError***. If adaptation is performed, place the new BIR into LPAdaptedBIR and ***SetLastError*** to

HAAPI-ADAPTEDBIR. In the event that adaptation is not supported, ***SetLastError*** to HAAPI_ADAPTATIONNOTSUPPORTED.

Remarks:

If a Biometric Identifier Record is passed in as the parameter lpStoredBIR, and the match is successful, ***LiveVerifyFn*** may attempt to adapt the enrolled BIR with information taken from the sample BIR. (It is up to the BSP vendor whether or not to provide adaptation.) If an adaptation is performed, the lpBioData member of the Biometric Identification Data structure must point to a valid address upon return from the function. Also, upon a successful match, set the extended error to HAAPI_ADAPTEDBIR.

In the event of an adaptation, this function allocates the data pointed to by the lpBioData member of the Biometric Identification Data structure. It is the application programmer's responsibility to free this allocated memory using the ***FreeFn*** function.

The ulSize member of the Biometric Identification Record structure indicates the size of the data allocated.

The Data stored in the Biometric Identification Record structure can be a concatenation of multiple identifiers.

Setting of verification thresholds or returning of verification matching scores may be provided by individual biometric technology vendors. If so, access is provided using the exported ***InformationFn*** function.

5.9 Free Memory

The free interface allows the developer to release the memory allocated by some of the Authentication functions. Since there are various ways to allocate memory, and many incompatibilities between compiler vendor's C runtime libraries, the vendor must provide the free function.

A Vendor **must** provide an exported free function and specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\FreeFunctionName**

5.9.1 Exported Free Function

HAAPIERROR FreeFn(LPVOID lpData**);**

Description:

This function frees the memory passed to it in lpData.

Parameters:

lpData – a void pointer to a block of memory.

Return Value:

Upon successful execution of a free, ***FreeFn*** returns HAAPI_NOERROR. If the function fails return the appropriate HAAPI_ERROR via a call the Win32 API, ***SetLastError***.

5.10 Technology Specific Parameters

Any biometric technology/product specific parameters are the responsibility of the vendor. It is suggested that such parameters be registered at the system level (registry settings) at the time of installation of the vendor's software and documented in the product literature. The Information function may be used for application program access to get or set these parameters.

The information interface is provided to make available certain valuable information to the HA-API compliant application developer, and allowing them (in certain cases) to alter the information. This function must support a minimal set of values. Additional values can be supported, depending on the desires of the vendor to make information available.

A Vendor **must** provide an exported information function and specify the name as a string value in the registry under the key:

HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName******TechnologyName***\InformationFunctionName**

5.10.1 Exported Information Function

HAAPIERROR InformationFn(long lRequestedInfo,
LPVOID lpData,
DWORD cbSize)

Description:

This function provides an interface for the application developer to get and set information specific to a Biometric Technology. Altering information using this function can adversely affect the performance and matching of a particular Biometric Technology. Care should be exercised when using this function. It must also be noted that vendors are only required to support a minimal set of constants for the requested information parameter. Any parameters used outside of this set are considered specific to the vendor and may not work across multiple vendor applications.

Parameters:

lRequesetInfo – a long integer constant defining the requested information being retrieved or set.

lpData – a pointer to the variable that contains the data to be set or will hold the data being retrieved.

cbSize – the size, in bytes, of lpData.

Return Value:

Upon successful capture, ***InformationFn*** returns HAAPI_NOERROR. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***.

Remarks:

All vendors must support the following constants:

Constant	Data Type for lpData
HAAPI_MAJORVERSION	long (read only)
HAAPI_MINORVERSION	long (read only)
HAAPI_BUILDVERSION	long (read only)
HAAPI_BUILDDATE	long (read only – julian date)
HAAPI_VENDORNAME	LPTSTR (read only)
HAAPI_TECHNAME	LPTSTR (read only)

Variable	Data Type for lpData
HAAPI_GET_LAST_SCORE	Scoring Record (read only)
HAAPI_SET_SYSTHRESHOLD	Threshold Record (write)
HAAPI_GET_SYSTHRESHOLD	Threshold Record (read)

Notes: 1) 'read only' is from the perspective of the application.
2) Variable definitions are found in Section 6.0, Structures.

Vendors may provide additional constants/variables that can be read/write. It is important to remember that these constants will be specific to that vendor.

This function allocates data pointed to by lpData. The application programmer will free this allocated memory through the ***FreeFn*** function.

Under Windows NT, any strings returned by this function are Wide Character (UNICODE) strings. If UNICODE is not defined in your project, use the Win32 API WideCharToMultiByte to convert the string to ANSI Code Page.

Under Windows 95, any strings returned by this function are ANSI strings.

5.11 Standard Biometric Properties Screen

The standard biometric properties screen should provide an interface to the user to alter specific settings relating to a biometric.

A Vendor may provide an exported biometric properties function. If provided, the vendor must specify the name as a string value in the registry under the key:

**HKEY_LOCAL_MACHINE\SOFTWARE\HAAPI\Vendors*VendorName*\
TechnologyName\BioPropertiesFunctionName**

5.10.1 Exported Biometric Properties Function

HAAPIERROR BioPropertiesFn(LPSCREENATTRIBS lpScreenAttribs**)**

Description:

This function displays a biometric properties dialog box. This dialog box is used to alter specific settings to the biometric.

Parameters:

lpScreenAttribs – a pointer to a Screen Attributes Data structure. This contains information for dialog box placement. If this parameter is NULL, the vendor default screen placement values will be used by the vendor module. For more information on Screen Attributes Data structures, see ***Section 6.0, Structures***.

NOTE: For applications that do not include a GUI (e.g., embedded applications) lpScreenAttribs will be set to NULL.

Return Value:

Upon successful capture, ***BioPropertiesFn*** returns HAAPI_NOERROR. If this function fails, return the appropriate HAAPI_ERROR via call to the Win32 API, ***SetLastError***. If this function is not supported ***SetLastError*** to HAAPI_NOBIOPROPERTIESPAGE.

6.0 Structures

The data structures herein defined have been designed to be as flexible as possible, allowing the biometric vendor to store whatever information is needed, without unnecessary constraints. For example, the biometric data structures may contain a single biometric sample or may contain multiple samples.

In the simplest case, the raw biometric data structure will be used to store the raw, yet unprocessed data resulting from a capture, and the biometric identifier record will be used to hold the data resulting from a processing operation. However, in order to support a wide range of process flow possibilities and biometric templates (models), these structures can be used to store any combination of data necessary to facilitate subsequent matching. It is the responsibility of the biometric technology provider (BSP) to fill this data structure with the data needed and in the format needed, and to be able to extract this data when it is needed.

It is recommended that the vendor include header information within the data structure to facilitate its identification. We strongly encourage BSPs to document standard data types and formats within the raw biometric data or BIR structures, to facilitate and allow applications to make use of this data. Specifically, if raw biometric data is stored in BMP, TIFF, JPEG, GIF, WAV, AU or other standard formats, documentation should allow for the extraction and display of individual records.

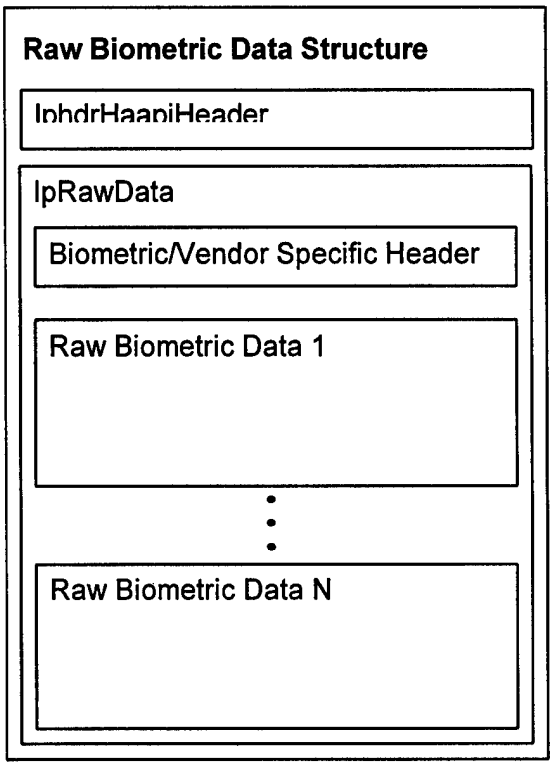
6.1 Raw Biometric Data Structure

The Raw Biometric Data Structure is generally used to hold the unprocessed data captured by a biometric capture device. For example, with finger imaging or facial recognition, this may be one or more bit mapped images; for speaker verification, it may be sampled audio waveforms. However, depending on the biometric vendor/technology, it may be used to hold data that has undergone some degree of processing in lieu of or in addition to the raw data. The actual contents of the structure should be included in the BSP documentation.

Most applications should not need to manipulate the contents of this data structure; however, should this be required, refer to the BSP documentation to interpret Biometric/Vendor Specific Header and to determine data type and format.

```
typedef struct tagRBD{
```

```
    LPHDR lphdrHaapiHeader;
    LPVOID lpRawData;
} RAWBIODATA, *LPRAWBIODATA, *PRAWBIODATA;
```



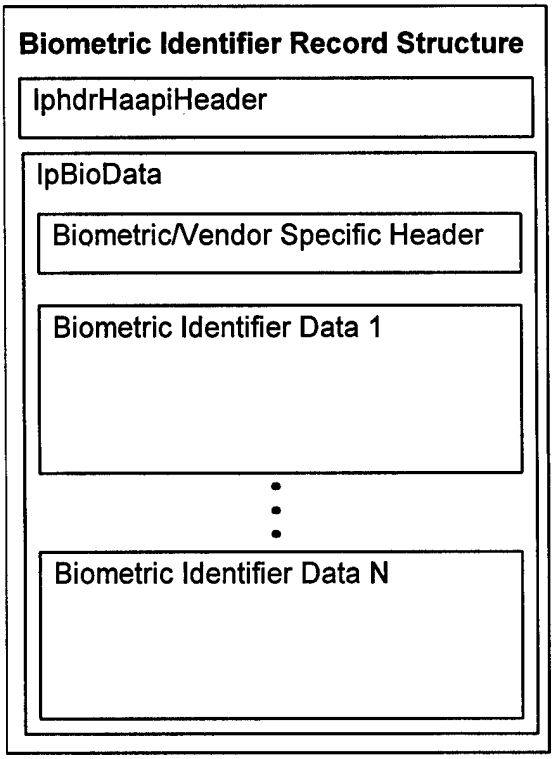
Member	Description
lphdrHaapiHeader	A pointer to the generic HAAPI header for this Raw Biometric Data structure.
lpBioData	A pointer to the Raw Biometric Data.

6.2 Biometric Identifier Record Structure

The Biometric Identifier Record (BIR) Data Structure is generally used to hold the processed biometric data. For example, with finger imaging it may be the extracted minutiae points; for facial recognition, the facial features; or for speaker verification, the audio characteristics. However, depending on the biometric vendor/technology, it may be used to hold raw data or data that has undergone some degree of processing in lieu of or in addition to the identifier data. The actual contents of the structure should be included in the BSP documentation.

Most applications should not need to manipulate the contents of this data structure; however, should this be required, refer to the BSP documentation to interpret Biometric/Vendor Specific Header and to determine data type and format.

```
typedef struct tagBIR{
        LPHDR lphdrHaapiHeader;
        LPVOID lpBioData;
} BIR, *LPBIR, *PBIR;
```

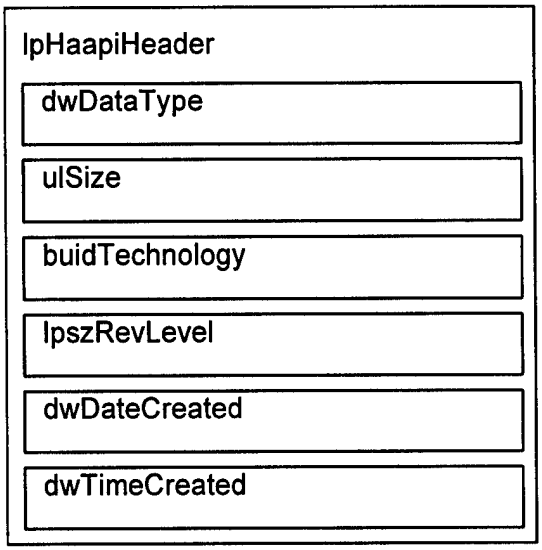


Member	Description
lphdrHaapiHeader	A pointer to the generic HAAPI header for this BIR.
lpBioData	A pointer to the Biometric Identifier Data.

6.3 HA-API Header Structure

The HaapiHeader is included with the Raw Biometric Data and Biometric Identifier Data Structures, to provide standard header information within each to facilitate import/export of this data.

```
typedef struct tagHH{
        DWORD dwDataType;
        ULONG ulSize;
        BUID buidTechnology;
        LPTSTR lpszRevLevel;
        DWORD dwDateCreated;
        DWORD dwTimeCreated;
} HDR, *LPHDR, *PHDR;
```

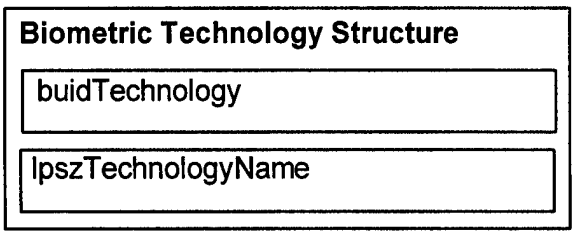


Member	Description
dwDataType	Flag Indicating if the data contained in the structure is raw biometric data, processed biometric identifier data, or both. Valid values are: RAW_TYPE BIR_TYPE RAW_TYPE | BIR_TYPE
ulSize	Unsigned long value indicating the size, in bytes, of the data following the HAAPI header.
buidTechnology	The BUID of the BSP which generated the data stored in as a GUID structure.
lpszRevLevel	A Null terminated string indicating the Rev level of the BSP which generated the data stored in the structure. Under Windows NT, this string will always be UNICODE. Under Windows 95, this string will always be ANSI.
dwDateCreated	The date on which the data structure was created.
dwTimeCreated	The time of day when the data structure was created.

6.4 Biometric Technology Structure

The Biometric Technology Structure is used to identify an available biometric technology by its Biometric Unique Identifier (BUID) and name (which are registered when the BSP is installed).

```
typedef struct tagBT{
        BUID buidTechnology;
        LPTSTR lpszTechnologyName;
} BIOTECH, *LPBIOTECH, *PBIOTECH;
```

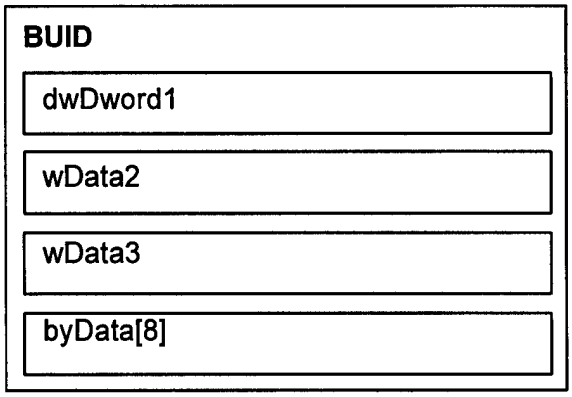


Member	Description
buidTechnology	A structure containing the BUID data for a particular BSP.
lpszTechnologyName	A pointer to a string indicating the name of a technology. Under Windows NT, this string will always be UNICODE. Under Windows 95, this string will always be ANSI.

6.5 BUID Structure

The Unique Biometric Identifier (BUID) is a Global Unique Identifier (GUID) generated by a vendor. This BUID is used to identify an available biometric technology.

```
typedef struct tagBUID{
        DWORD dwData1;
        WORD wData2;
        WORD wData3;
        BYTE byData4[ 8 ];
} BUID, *LPBUID, *PBUID;
```

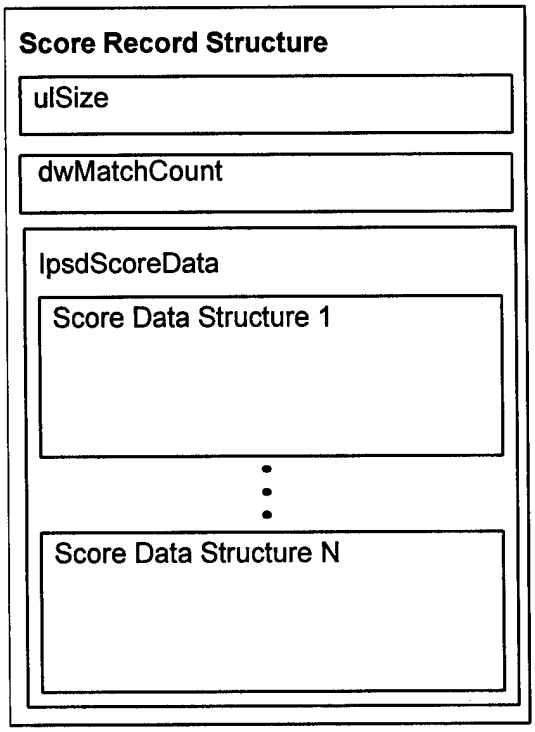


Member	Description
dwDword1	The first DWORD value of the BUID.
wData2	The second WORD value of the BUID.
wData3	The third WORD value of the BUID.
byData4	An array of 8 bytes, comprising the fourth data element of the BUID

6.6 Scoring Record Structure

The Scoring Record is used to provide information to the application regarding the preceding matching function. It returns the matching score calculated and threshold setting used during the last performed match. If the match was performed against more than one record, or if the score consists of multiple values, each set of scores will be returned. If an individual/subject threshold was used for the match, it is also returned.

```
typedef struct tagSR{
        ULONG ulSize;
        DWORD dwMatchCount;
        LPSD lpsdScoreData
} SIR, *LPSIR, *PSIR;
```

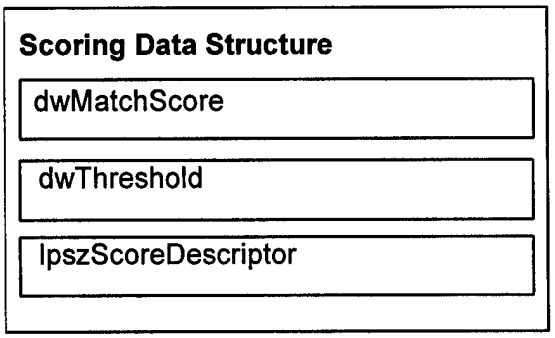


Member	Description
ulSize	Size of lpsdScoreData in bytes.
dwMatchCount	The number of structures included in lpsdScoreData array.
lpsdScoreData	A pointer to an array of Score Data structures.

6.7 Scoring Data Structure

The Scoring Data Structure contains the data for a particular score. It contains the matching score calculated and threshold setting used during the last performed match. It also contains a descriptor of the score.

```
typedef struct tagSD{
        DWORD dwMatchScore;
        DWORD dwThreshold;
        LPTSTR lpszScoreDescriptor;
} SD, *LPSD, *PSD;
```



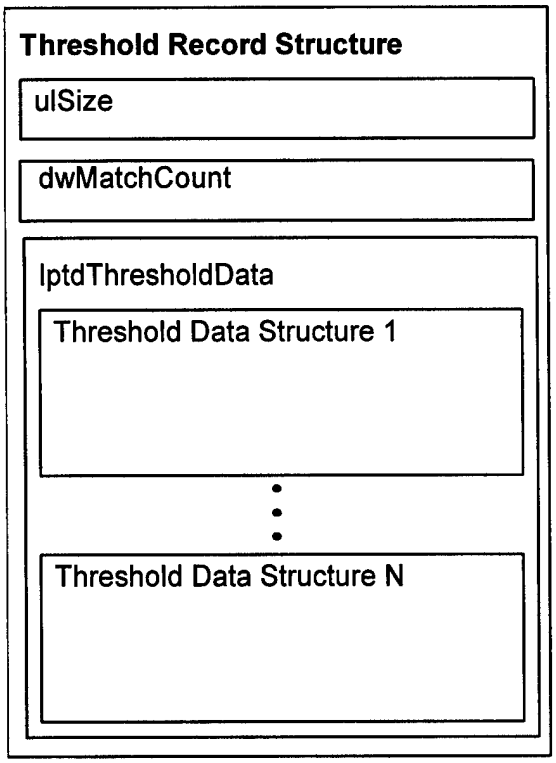
Member	Description
dwMatchScore	The value of the match score for the last verification match performed. This is a value between 0 and 100 (continuous), with 100 being the highest.
dwThreshold	The threshold value used with the associated match score. This is a value between 0 and 100 (continuous), with 100 being the highest.
lpszMatchDescriptor	A pointer to a NULL terminated string value, which can contain amplification data regarding the associated match score. Under Windows NT, this is always a UNICODE string. Under Windows 95, this is always an ANSI string.

See Appendix F for guidance regarding setting and use of match scores and thresholds.

6.8 Threshold Record Structure

The Threshold Record is used to get and set the system level matching threshold to be used in subsequent matches. It contains either a single value or a set of threshold settings.

```
typedef struct tagSR{
        ULONG ulSize;
        DWORD dwMatchCount;
        LPTD lptdThresholdData
} THREC, *LPTHREC, *PTHREC;
```

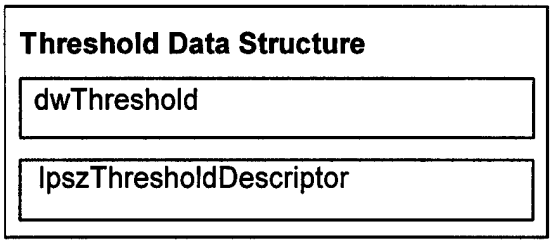


Member	Description
ulSize	Size of lptdThresholdData in bytes.
dwMatchCount	The number of structures included in lptdThresholdData array.
lptdThresholdData	A pointer to an array of Score Data structures.

6.9 Threshold Data Structure

The Threshold Data Structure contains the data for a particular threshold. It contains the threshold setting and descriptor.

```
typedef struct tagTD{
        DWORD dwThreshold;
        LPTSTR lpszThresholdDescriptor;
} TD, *LPTD, *PTD;
```



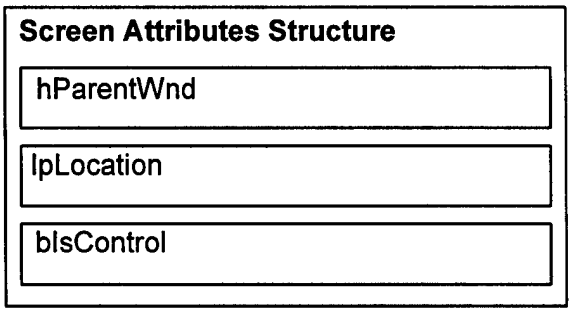
Member	Description
dwThreshold	The value used to determine if a match has occurred. This is a value between 0 and 100 (continuous), with 100 being the highest.
lpszMatchDescriptor	A pointer to a NULL terminated string value, which can contain amplification data regarding the threshold value. Under Windows NT, this is always a UNICODE string. Under Windows 95, this is always an ANSI string.

See Appendix F for guidance regarding setting and use of match scores and thresholds.

6.10 Screen Attributes Structure

The screen attributes structure provides information to the vendor module for placement of a dialog box. The first member, hParentWnd, indicates the parent window to whom the dialog box belongs. The second member, lpLocation, is a pointer to an X,Y pair that gives the screen location of the dialog box, in screen coordinates. The final member, bIsControl, if TRUE, indicates to the vendor module that the dialog box should be displayed as a child control of the parent window, hParentWnd (using the DS_CONTROL style). If this is the case, lpLocation, is the location in Client Coordinates.

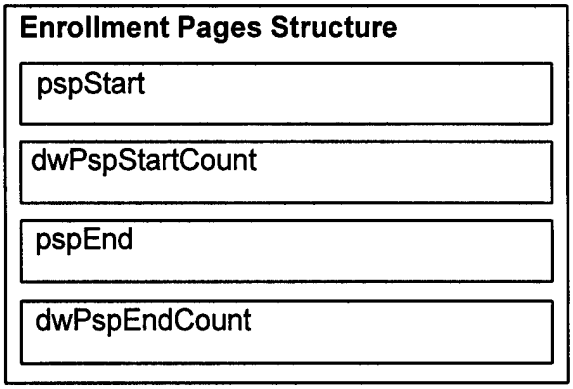
```
typedef struct tagSA{
        HWND hParentWnd;
        LPPOINT lpLocation;
        BOOL bIsControl;
} SCREENATTRIBS, *LPSCREENATTRIBS, *PSCREENATTRIBS;
```



6.11 Enrollment Pages Structure

The enrollment pages structure provides information to the vendor module about any additional enrollment wizard pages needed for the application. Enrollment pages can be added to the beginning or end of the enrollment wizard. The first member, pspStart, is an array of PROPSHEETPAGE structures. These structures provide the enrollment wizard with information about the pages to be added to the beginning of the wizard. The second member, dwPspStartCount, indicates the number of PROPSHEETPAGE structures in the pspStart array. The third member, pspEnd, is an array of PROPSHEETPAGE structures. These structures provide the enrollment wizard with information about the pages to be added to the end of the wizard. The forth member, dwPspEndCount, indicates the number of PROPSHEETPAGE structures in the pspEnd array. Additional information on the PROPSHEETPAGE structure can be found in the Microsoft Platform SDK.

```
typedef struct tagEP{
        PROPSHEETPAGE *pspStart;
        DWORD dwPspStartCount;
        PROPSHEETPAGE *pspEnd;
        DWORD dwPspEndCount;
} BIOTECH, *LPBIOTECH, *PBIOTECH;
```

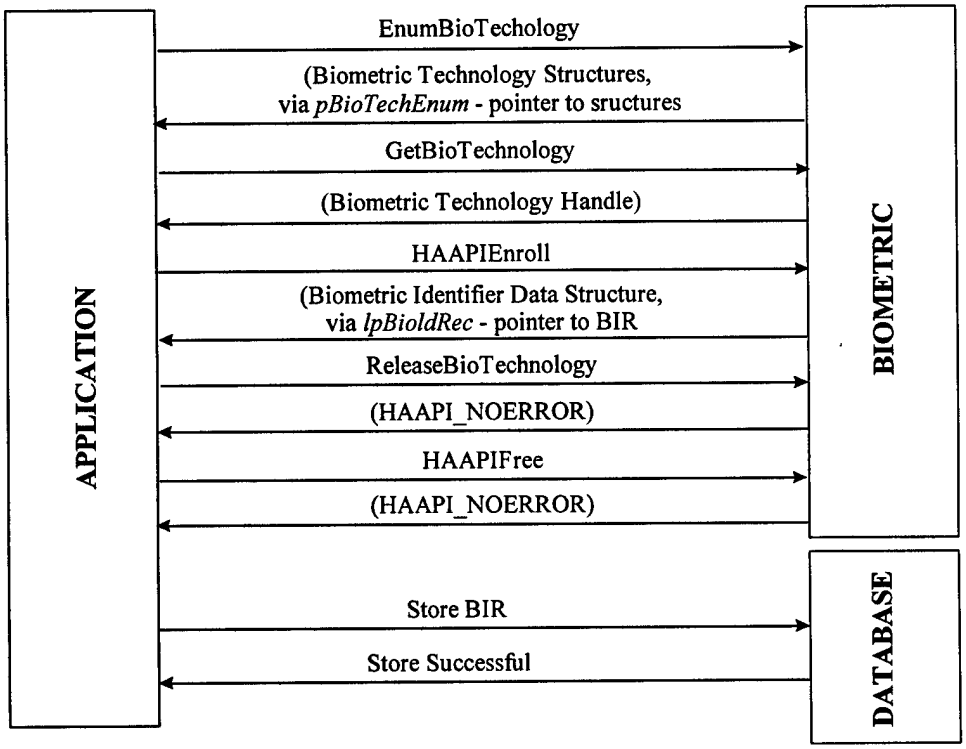


7.0 Sample message flows

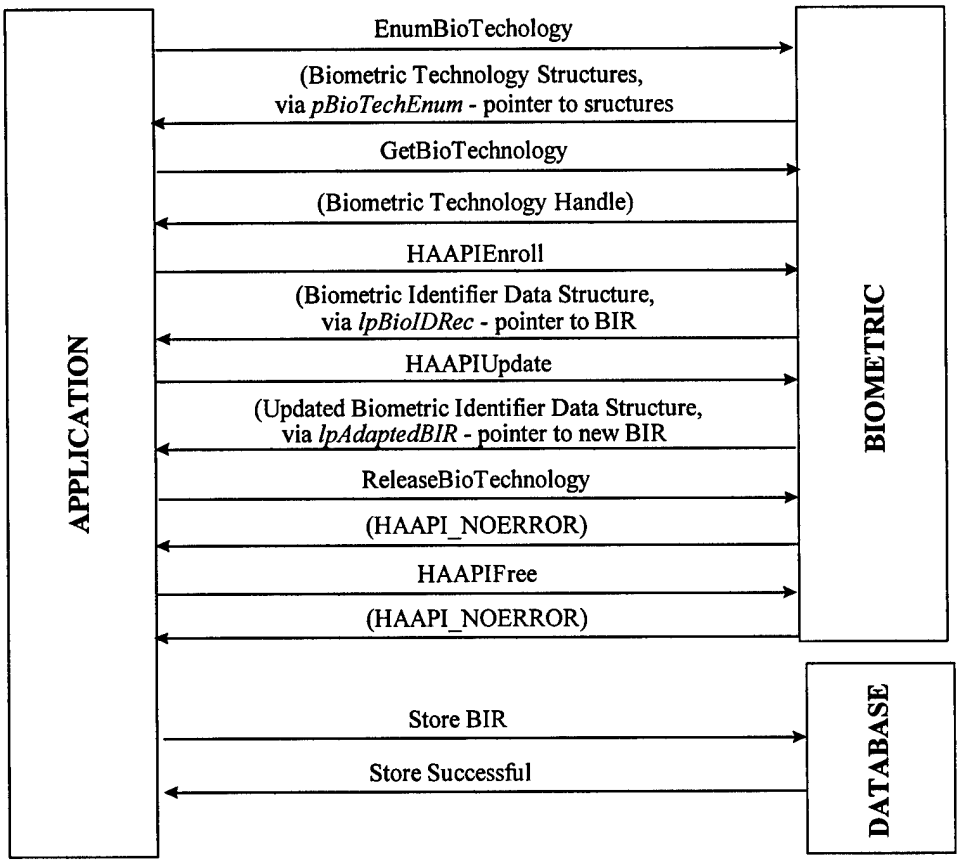
The following sections depict the sequence of HA-API calls and returned data to perform the Enrollment and Verification top-level functions.

7.1 Enrollment

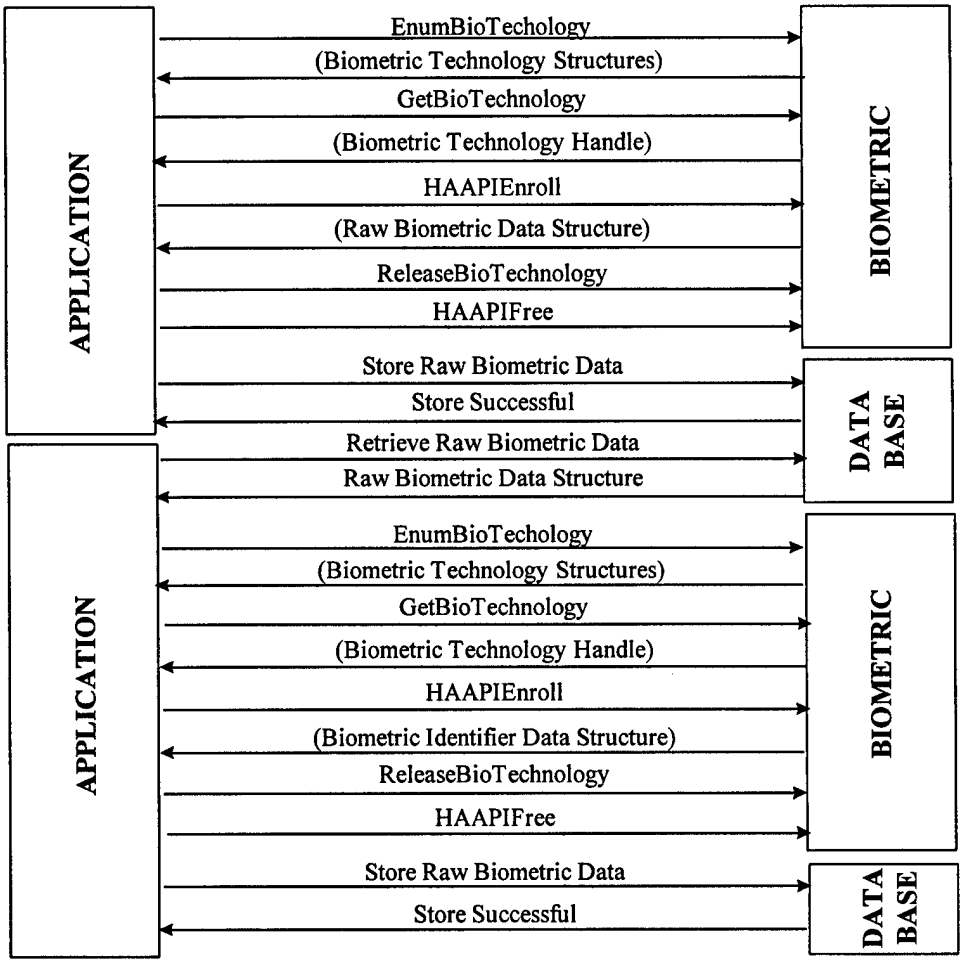
7.1.1 New Enrollment



7.1.2 Update Enrollment

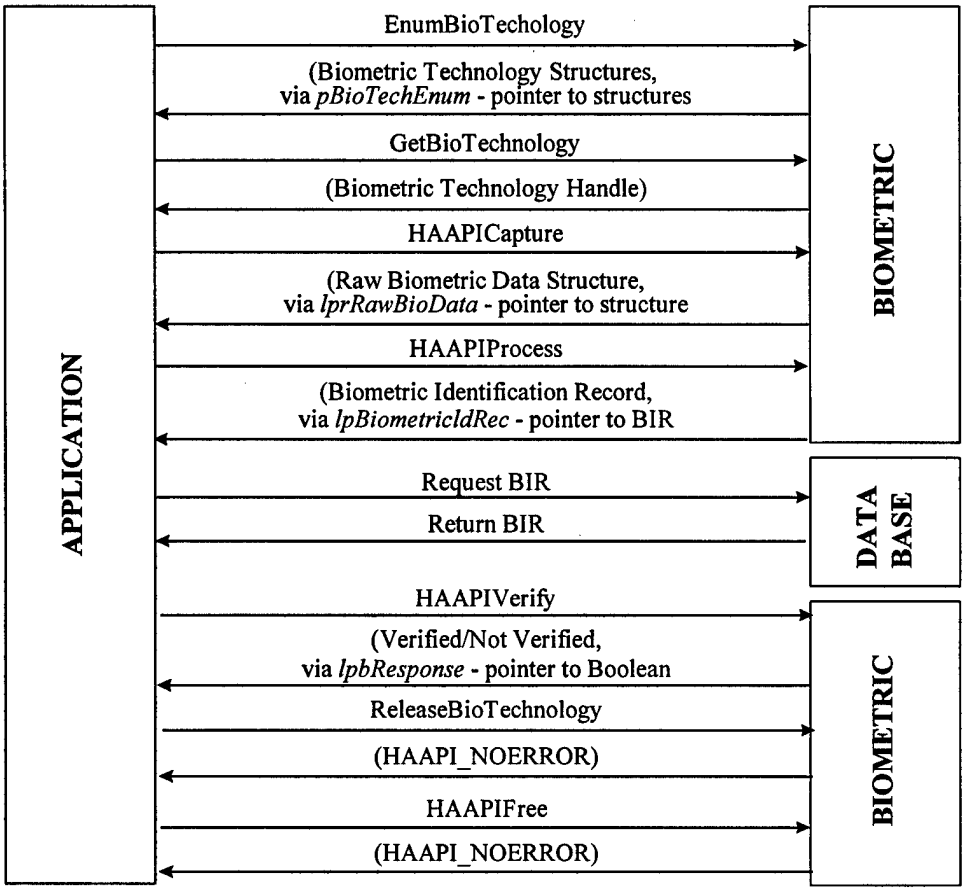


7.1.3 Batch Enrollment

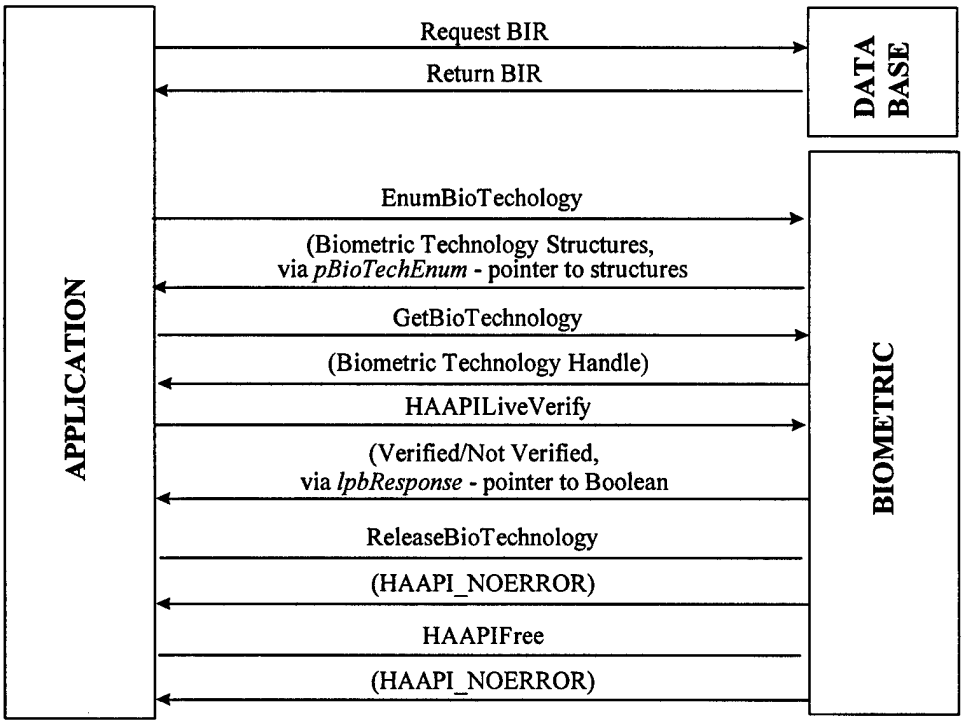


7.2 Verification

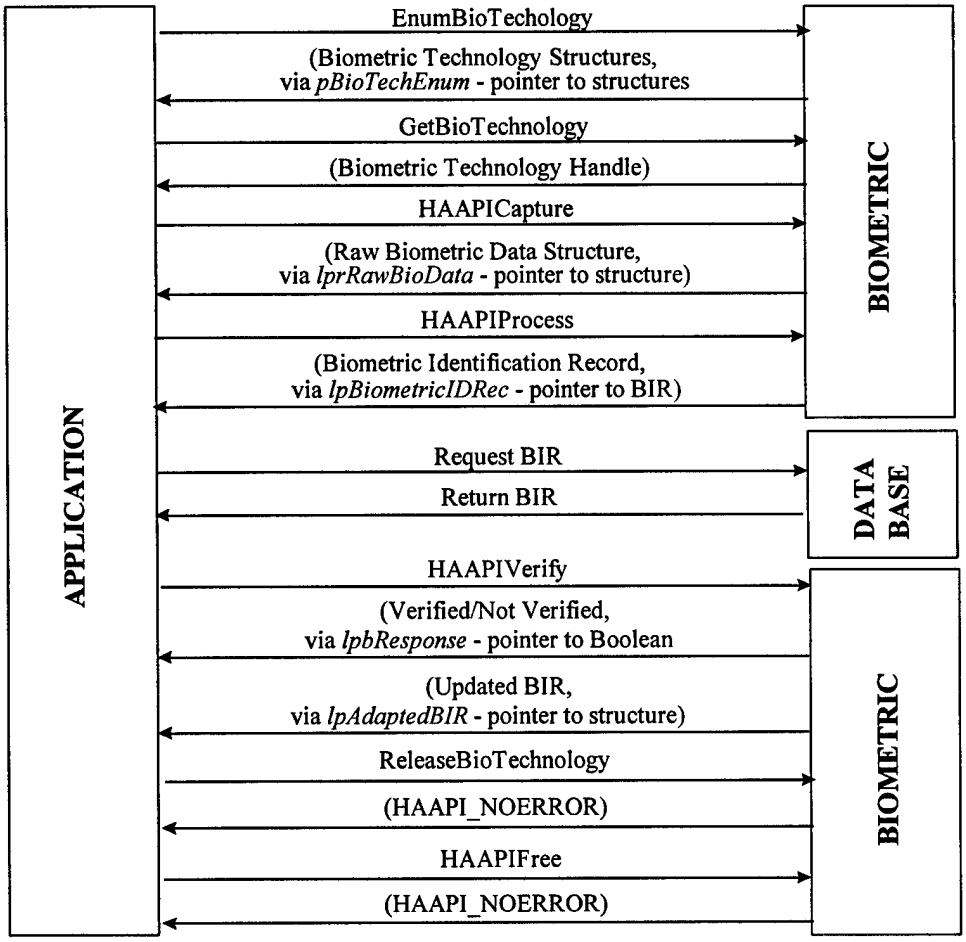
7.2.1 Normal Verify



7.2.2 Live Verify



7.2.3 Verify with Update



8.0 References

Federal Information Processing Standards (FIPS) Publication 190, Guideline for the Use of Advanced Authentication Technology Alternatives, National Institute of Standards and Technology, 28 Sep 94

DoD 5200.28.STD, Department of Defense Trusted Computer System Evaluation Criteria (aka, the "Orange Book"), Dec 1985

The Windows Interface Guidelines for Software Design

9.0 Glossary

9.1 Acronyms

API	Application Program Interface
BIR	Biometric Identifier Record
BSP	Biometric Service Provider
BUID	Unique Biometric Identifier
DoD	Department of Defense
FIPS	Federal Information Processing Standard
GUI	Graphical User Interface
HA-API	Human Authentication - Application Program Interface
I&A	Identification and Authentication
ID	Identification, Identifier
INFOSEC	Information Security
LAN	Local Area Network
PIN	Personal Identification Number
NRI	The National Registry Inc.
PDC	Primary Domain Controller
SA	System Administrator
SAF/NT	Secure Authentication Facility for Windows NT
SPI	Service Provider Interface
SSA	System Security Administrator
TCP/IP	Transmission Control Protocol

9.2 Definitions

Term	Definition
Authentication	A means of positive identification used to verify authorization to access information or resources.
Biometric	In access control, automated methods of verifying or recognizing a person based upon a physical or behavioral characteristic. Biometric techniques may be classified on the basis of some passive attribute of an individual (e.g., fingerprint, eye retina pattern, speech pattern) or some unique manner in which an individual performs a task (e.g., writing a signature, typing).
Enrollment	Entry of biometric and other user information into an authentication database.
False Accept	When a sample biometric is incorrectly matched to a file sample.
False Reject	When a sample biometric is incorrectly determined not to match a file sample.
Identification	One-to-many matching of a single biometric sample set against a database of samples, to determine which, if any, it matches.
Information Security	The result of any system of policies/procedures and mechanisms for protecting from unauthorized disclosure, information whose protection is authorized.
Matching	Comparison of one biometric sample to another to determine if they belong to the same person.
Multiple Factor Authentication	Using more than one biometric or other mechanism in combination to verify the identity of a user.
Password	A protected word, phrase, or string of characters that identifies or authenticates a user for access to a specific resource, such as a system, process, or data set.
Template	The information extracted from captured raw biometric data that is stored and used in the comparison/matching process.
Threshold	A value either above or below which a match is determined to exist. (May also be used to determine if the quality of a captured biometric is acceptable or not.)
Token	A physical object, unique to a user, whose possession may be used to prove the identity of a human.
Verification	One-to-one matching of one biometric sample set against another.

Appendix A - Function Prototypes

```
HBT WINAPI CALLBACK GetBioTechnology(BUID buidBioTechnology);
HAAPIERROR WINAPI CALLBACK ReleaseBioTechnology(HBT hbtBioTechnology);
HAAPIERROR WINAPI CALLBACK EnumBioTechnology(
                        LPBIOTECH pBioTechEnum,
                        DWORD dwBuf,
                        LPDWORD pdwNeeded,
                        LPDWORD pdwReturned)

HAAPIERROR WINAPI CALLBACK HAAPIProcess(HBT hbtBioTechnology,
                        LPRAWBIO lpRawBioData,
                        LPBIR lpBiometricIdRec);
HAAPIERROR WINAPI CALLBACK HAAPIVerify(HBT hbtBioTechnology,
                        LPBIR lpSampleBIR,
                        LPBIR lpStoredBIR,
                        LPBIR lpAdaptedBIR,
                        LPBOOL lpbResponse);
HAAPIERROR WINAPI CALLBACK HAAPICapture(HBT hbtBioTechnology,
                        LPRAWBIODATA lpRawBioData);
HAAPIERROR WINAPI CALLBACK HAAPIEnroll(HBT hbtBioTechnology,
                        DWORD EnrollType,
                        LPSCREENATTRIBS lpScreenAttribs,
                        LPRAWBIODATA lpRawBioData,
                        LPBIR lpNewBioIDRec,
                        LPENROLLMENTPAGES lpPages);
HAAPIERROR WINAPI CALLBACK HAAPIUpdate(HBT hbtBioTechnology,
                        LPRAWBIODATA lpOldRawBioData,
                        LPRAWBIODATA lpNewRawBioData,
                        LPBIR lpStoredBioIDRec,
                        LPBIR lpNewBioIDRec)
HAAPIERROR WINAPI CALLBACK HAAPILiveVerify(HBT hbtBioTechnology,
                        LPBIR lpStoredBIR,
                        LPBIR lpAdaptedBIR,
                        int iTimeout,
                        LPBOOL lpbResponse)
```

```
HAAPIERROR WINAPI HAAPIFree(HBT hbtBioTechnology,
                            LPSCREENATTRIBS lpScreenAttribs);
HAAPIERROR WINAPI CALLBACK HAAPIInformation(HBT hbtBioTechnology,
                            long lVendorInfo,
                            LPVOID lpData,
                            DWORD cbSize);
HAAPIERROR WINAPI HAAPIBioProperties(HBT hbtBioTechnology,
                            LPSCREENATTRIBS lpScreenAttribs)
```

Appendix B - Defines

Types:

```
#define     HBT                 DWORD
#define     HAAPIERROR          long
#define     HAAPI_NOTIMEOUT     0x0000

#define HAAPI_ALL_TECH          0x0000
#define HAAPI_FINGER_TECH       BIO_ALL_TECH + 0x0001
#define     HAAPI_SPEECH_TECH   BIO_ALL_TECH + 0x0002
#define     HAAPI_FACIAL_TECH   BIO_ALL_TECH + 0x0003

#define     HAAPI_BASECONSTANT  0x0000
#define     HAAPI_MAJORVERSION  HAAPI_BASECONSTANT + 0x0001
#define     HAAPI_MINORVERSION  HAAPI_BASECONSTANT + 0x0002
#define     HAAPI_BUILDVERSION  HAAPI_BASECONSTANT + 0x0003
#define     HAAPI_BUILDDATE     HAAPI_BASECONSTANT + 0x0004
#define     HAAPI_VENDORNAME    HAAPI_BASECONSTANT + 0x0005
#define     HAAPI_TECHNAME      HAAPI_BASECONSTANT + 0x0006
#define     HAAPI_VENDORBASE    HAAPI_BASECONSTANT + 0x00FF

#defineRAW_TYPE                 0x0001
#defineBIR_TYPE                 0x0002

#defineENROLL_CAPTURE           0x0001
#defineENROLL_BIR               0x0002
#defineENROLL_RAW               0x0004
```

Appendix C - Enumerated Types

No Enumerated Types are currently defined.

Appendix D - C++ Classes

No C++ Classes are currently defined

Appendix E - Error Codes, BUIDs and GUIDs

As error codes are added to this list, they should be included in the error header file. The error code scheme is as follows: All functions return either HAAPI_NOERROR or HAAPI_ERROR. Upon error conditions, the vendor module must use the WIN32 function ***SetLastError*** to set a specific error code for the last event. This error code can then be retrieved by the application through a call to the WIN32 function ***GetLastError***. To further integrate this process into the WIN32 API, the vendor must provide a textual explanation of the error code, making the error code translatable through a call to ***FormatMessage***. For more information refer to the Microsoft Win32 documentation under Last-Error Code and Message Table Resources.

Error codes are the responsibility of the vendor. The only requirements for error codes are:

1. They do not overlap the pre-defined HAAPI error codes.
2. The 29th bit of the error code is set to 1.

The following error codes are returned from the HAAPI runtime DLL's. The vendor is not limited to using just these error codes.

Error Code	Value
HAAPI_NOERROR	0
HAAPI_ERROR	-1
HAAPI_BASEERROR	0x2F000000
HAAPI_INVALIDBUID	HAAPI_BASEERROR + 0x00000001
HAAPI_INVALIDPARAMETERS	HAAPI_BASEERROR + 0x00000002
HAAPI_REGISTRYERROR	HAAPI_BASEERROR + 0x00000003
HAAPI_MEMORYALLOCERROR	HAAPI_BASEERROR + 0x00000004
HAAPI_INVALIDMODULENAME	HAAPI_BASEERROR + 0x00000005
HAAPI_INVALIDTECHNOLOGYNAME	HAAPI_BASEERROR + 0x00000006
HAAPI_INVALIDENROLLMENTFNNAME	HAAPI_BASEERROR + 0x00000007
HAAPI_INVALIDCAPTUREFNNAME	HAAPI_BASEERROR + 0x00000008
HAAPI_INVALIDPROCESSFNNAME	HAAPI_BASEERROR + 0x00000009
HAAPI_INVALIDVERIFYFNNAME	HAAPI_BASEERROR + 0x0000000A

HAAPI_INVALIDLIVEVERIFYFNNAME	HAAPI_BASEERROR + 0x0000000B
HAAPI_INVALIDUPDATEFNNAME	HAAPI_BASEERROR + 0x0000000C
HAAPI_INVALIDINFOFNNAME	HAAPI_BASEERROR + 0x0000000D
HAAPI_INVALIDFREEFNNAME	HAAPI_BASEERROR + 0x0000000E
HAAPI_INVALIDBIOPROPFNNAME	HAAPI_BASEERROR + 0x0000000F
HAAPI_REGISTRYSUBSYSTEMFAILURE	HAAPI_BASEERROR + 0x00000010
HAAPI_USERCANCEL	HAAPI_BASEERROR + 0x00000011
HAAPI_GENERALERROR	HAAPI_BASEERROR + 0x00000012
HAAPI_ADAPTEDBIR	HAAPI_BASEERROR + 0x00000013
HAAPI_ADAPTATIONNOTSUPPORTED	HAAPI_BASEERROR + 0x00000014
HAAPI_NOBIOPROPERTIESPAGE	HAAPI_BASEERROR + 0x00000015
HAAPI_BUIDTABLEALREADYINIT	HAAPI_BASEERROR + 0x00000020
HAAPI_BUIDTABLENOTINIT	HAAPI_BASEERROR + 0x00000021
HAAPI_INVALIDFNTYPE	HAAPI_BASEERROR + 0x00000022
HAAPI_CORRUPTTABLEMANAGER	HAAPI_BASEERROR + 0x00000023
HAAPI_TABLEMANAGERFAILURE	HAAPI_BASEERROR + 0x00000024
HAAPI_UNABLETOLOADVENDORMODULE	HAAPI_BASEERROR + 0x00000025
HAAPI_UNABLETOUNLOADVENDORMODULE	HAAPI_BASEERROR + 0x00000026
HAAPI_VENDORMODULENOTPRESENT	HAAPI_BASEERROR + 0x00000027
HAAPI_MODULESUBSYSTEMFAILURE	HAAPI_BASEERROR + 0x00000030
HAPPI_MODULESUBSYSTEMCORRUPT	HAAPI_BASEERROR + 0x00000031
HAAPI_INVALIDCAPTUREFUNCTION	HAAPI_BASEERROR + 0x00000040
HAAPI_INVALIDENROLLFUNCTION	HAAPI_BASEERROR + 0x00000041
HAAPI_INVALIDVERIFYFUNCTION	HAAPI_BASEERROR + 0x00000042
HAAPI_INVALIDLIVEVERIFYFUNCTION	HAAPI_BASEERROR + 0x00000043
HAAPI_INVALIDPROCESSFUNCTION	HAAPI_BASEERROR + 0x00000044
HAAPI_INVALIDINFOFUNCTION	HAAPI_BASEERROR + 0x00000045
HAAPI_INVALIDFREEFUNCTION	HAAPI_BASEERROR + 0x00000046
HAAPI_INVALIDUPDATEFUNCTION	HAAPI_BASEERROR + 0x00000047
HAAPI_INVALIDBIOPROPFUNCTION	HAAPI_BASEERROR + 0x00000048
HAAPI_INVALIDOS	HAAPI_BASEERROR + 0x00000050
HAAPI_INVALIDHBT	HAAPI_BASEERROR + 0x00000051

Appendix F - Scores and Thresholds

The following discussion of scoring and thresholds is provided to emphasize the criticality of these values in a biometric system. In most cases, a biometric vendor's pre-set threshold is adequate and need not be changed, nor matching scores returned. In the rare situations where this is required, it is important to understand the effect of changing the pre-set threshold on the ultimate performance (i.e., matching accuracy) of the system.

When two identifiers are compared in order to determine whether or not they **match** (i.e., belong to the same person, or part of a person), an algorithm is used compare the two entities and return a **score**, which represents a probability that the two biometric characteristics are the same. This is not as simple as comparing two string values, where the comparison determines if an exact match exists or does not exist. Once a score has been calculated, it must be compared against a pre-set **threshold** value. If the score is above the threshold, a match is said to exist. If the score is below the threshold, a match is said not to exist.

The threshold value used determines the accuracy of the system in terms of error rates. There are two types of errors that can occur in a biometric system, a **false accept** (match) or a **false reject** (non-match).

A false accept occurs when two biometrics which should not match (belong to different people) are determined to match, (that is, the match score exceeded the threshold).

A false non-match occurs when two biometric which should match (belong the same person) are determined not to match (that is, the match score did not exceed the threshold). This can happen, for example, when a person places their finger incorrectly on a finger image scanner. This is generally corrected by a second attempt.

No biometric technology is 100% accurate, although many are close. The overall error rate is made up of both false accept and false reject errors. Generally, changing a matching threshold will cause one type of error to increase and the other to decrease. Depending on the application in which it is used, one type of error may be more acceptable than another. For example, biometrics for a physical access control to a nuclear weapons facility may be intolerant of false accepts, and therefore more tolerant of false rejects. A customer service application, on the other hand, may choose to accept a higher level of false matches in order to avoid falsely rejecting a customer.

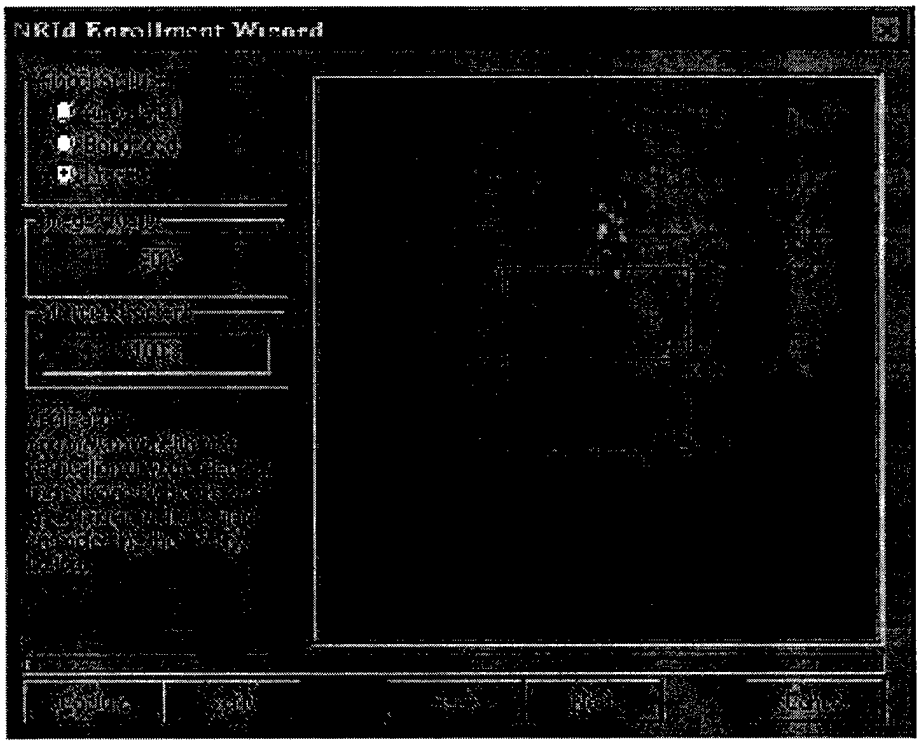
Within HA-API, an application is able to get and set thresholds and to request matching scores using the HAAPIInformation function. The scores/thresholds returned are defined to be a floating point value between 0 and 100. It is **very important** to realize that the values returned by a BSP are generally NOT LINEAR and VARY SIGNIFICANTLY from one BSP to another. It is not correct to compare in any way scores returned by two different BSPs. Nor is it generally appropriate to use the same threshold setting for two different BSPs, as the effect on the accuracy of each may be vastly different. Refer to the BSP vendor documentation to interpret the score/threshold data used by that specific BSP.

A threshold set incorrectly can cause a system to perform poorly. BSP vendors know their own technology better than anyone else can. They determine the default threshold setting, which is set upon installation. Applications that choose to alter this value do so **at their own risk**.

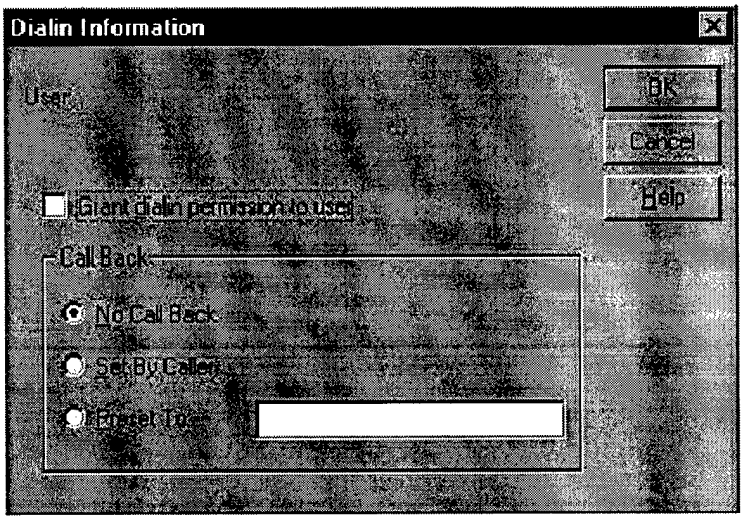
Appendix G - Sample Screens

The following screens are provided as examples of Windows GUI implementations. Screens provided by biometrics technology vendors for inclusion in the HA-API BSP are expected to employ similar mechanisms.

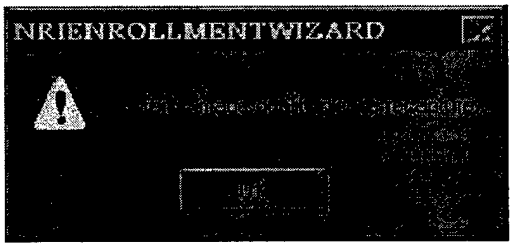
Enrollment Capture Screen

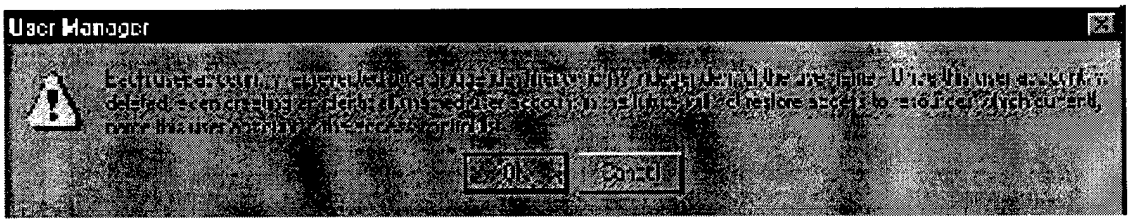


Dialog Box



Message Box





INTERNET DOCUMENT INFORMATION FORM

A . Report Title: Interface Specification Human Authentication-Application Program Interface (HA-API) Version 2.0

B. DATE Report Downloaded From the Internet 8/31/98

C. Report's Point of Contact: (Name, Organization, Address, Office Symbol, & Ph #): **U.S. Biometrics Consortium**
ATTN: Major John Colombi, PHD
C/0 NSA/R22, Suite 6516
9800 Savage Rd
Fort Meade, MD 20755-6516

D. Currently Applicable Classification Level: Unclassified

E. Distribution Statement A: Approved for Public Release

F. The foregoing information was compiled and provided by:
DTIC-OCA, Initials: UM **Preparation Date**: 9/31/98

The foregoing information should exactly correspond to the Title, Report Number, and the Date on the accompanying report document. If there are mismatches, or other questions, contact the above OCA Representative for resolution.