DEDICATION

To:

*My son, Felix Fuentes III, for giving me a fresh outlook at life,*

*And*

*My mother, for always being by my side.*
ABSTRACT

There is increasing demand world-wide, from government agencies and the private sector for cutting-edge biometric security technology that is difficult to breach. Some older tools, such as fingerprint, retina and iris scanning and facial recognition software have all been found to have flaws. However, reproducing a three-dimensional model of a human vein system is impossible to replicate. Mapping veins as a human barcode is the newest technology to hit the security world which has key benefits over older technologies. Vein map technology is distinctive because of its state-of-the-art sensors are only able to recognize vein patterns if hemoglobin is actively flowing through the person's veins. Additionally, each individual's vein map is unique, even in the case of identical twins. The combinations of these factors could give vein map authentication an edge over existing biometric identification products.

This project briefly describes the types of biometric systems in place today, then goes into detail on vein-map authentication, and finally concludes with a software product implemented using Microsoft’s Visual Studio 2005 programming environment which interfaces with Fujitsu’s newly released vein map scanner.
# TABLE OF CONTENTS

Abstract ................................................................................................................... iii

Table of Contents ................................................................................................... iv

List of Figures ........................................................................................................ vi

Appendices ............................................................................................................ vii

1. Background and Rationale .................................................................................. 1

1.1 Introduction to Biometrics .............................................................................. 1

1.2 History of Biometrics ...................................................................................... 1

1.3 Some Modern Biometric Systems ................................................................. 3

1.3.1 Fingerprint Scanning ................................................................................ 3

1.3.2 Iris Recognition ......................................................................................... 4

1.3.3 Facial Recognition ..................................................................................... 6

1.4 Benefits of biometrics .................................................................................... 10

1.5 Issues in Biometrics ....................................................................................... 10

1.5.1 Hygiene ..................................................................................................... 10

1.5.2 Religious Objectives ............................................................................... 11

1.5.3 Misuse of Data ......................................................................................... 11

1.6 Vein Map Authentication ................................................................................. 12

2. Biometrics and Vein Map Authentication ....................................................... 13

2.1 Registration Tab .......................................................................................... 14

2.2 Authentication Tab ....................................................................................... 14

2.3 Door Control Tab ......................................................................................... 15

3. Proposed System Design and Implementation ............................................... 17
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Minutiae</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Other Fingerprint Characteristics</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Iris Diagram</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Iris Structure</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>Standard Eigenfaces: Feature vectors are derived using eigenfaces</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Example of six classes using LDA</td>
<td>8</td>
</tr>
<tr>
<td>1.7</td>
<td>Elastic Bunch Map Graphing</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Registration Tab Page</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Authentication Tab Page</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Door Switch Tab Page</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Proposed System Architecture</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Sample Vein Map Table in SQL Server 2005</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Registration Data Flow</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Authentication Data Flow</td>
<td>27</td>
</tr>
<tr>
<td>3.5</td>
<td>PalmSecure Specification</td>
<td>28</td>
</tr>
<tr>
<td>3.6</td>
<td>Vein Map after near infrared light Penetration</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>SEAIO class as standalone application</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>FVM class as standalone application</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>iCam class as standalone application</td>
<td>31</td>
</tr>
<tr>
<td>4.4</td>
<td>Windows XP Performance Report during Authentication</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>Windows Vista Performance Report during Authentication</td>
<td>34</td>
</tr>
</tbody>
</table>
### APPENDICES

1.2 Source Code........................................................................................................40
  1.2.1 FVM Class Definition.................................................................40
  1.2.2 iCam Class Definition...............................................................48
  1.2.3 VMA Class Definition...............................................................51
1.3 Copy Rights.......................................................................................................55
1. BACKGROUND AND RATIONALE

Biometrics are automated methods of recognizing a person based on a physiological or behavioral characteristic. Among the features measured are the: face, fingerprints, hand geometry, handwriting, iris, retinal, vein, and voice. Biometric technologies are becoming the foundation of an extensive array of highly secure identification and personal verification solutions. As the level of security breaches and transaction fraud increases, the need for highly secure identification and personal verification technologies is becoming apparent.

1.1 Introduction to Biometrics

The term “biometrics” is derived from the words “bio” (life) and “metrics” (to measure). Automated biometric systems have only become available over the last few decades, due to significant advances in the field of computer processing. Many of these new automated techniques, however, are based on ideas that were originally conceived hundreds of years ago [Wikipedia 2006].

Biometric-based solutions are able to provide for confidential financial transactions and personal data privacy. The need for biometrics can be found in federal, state and local governments, military, and in commercial applications. Enterprise-wide network security infrastructures, government IDs, secure electronic banking, investing and other financial transactions, retail sales, law enforcement, and health and social services are already benefiting from these technologies [findBiometrics 2006].

1.2 History of Biometrics

Possibly the first known example of biometrics in practice was a form of finger printing being used in China in the 14th century, as reported by explorer Joao de Barros.
He wrote that the Chinese merchants were stamping children's palm prints and footprints on paper with ink to distinguish the young children from one another. This is one of the earliest known cases of biometrics in use and is still being used today [Biometrics Insight, 2006].

Elsewhere in the world up until the late 1800s, identification largely relied upon "photographic memory." In the 1890s, an anthropologist and police desk clerk in Paris named Alphonse Bertillon sought to fix the problem of identifying convicted criminals and turned biometrics into a distinct field of study. He developed a method of multiple body measurements which got named after him (Bertillonage) [Biometrics Insight, 2006]. His system was used by police authorities throughout the world, until it quickly faded when it was discovered that some people shared the same measurements and based on the measurements alone, two people could get treated as one. After the failure of Bertillonage, the police started using fingerprinting, which was developed by Richard Edward Henry of Scotland Yard [VideoSurveillanceGuide, 2006], essentially reverting to the same methods used by the Chinese for years.

In the past three decades biometrics has moved from a single method (fingerprinting) to more than ten discreet methods. Companies involved with new methods number in the hundreds and continue to improve their methods as the technology available to them advances. Prices for the hardware required continue to fall making systems more feasible for low and mid-level budgets. As the industry grows however, so does the public concern over privacy issues. Laws and regulations continue to be drafted and standards are beginning to be developed.
1.3 Some Modern Biometric Systems

To date, popular biometric authentication systems include, fingerprint identification, retina and iris scan, face recognition, and voice analysis. The problem with each system is that most can be breached easily or intrude upon the rights of individuals or both.

1.3.1 Fingerprint Scanning

Fingerprint identification is one of the most well-known and publicized biometrics. Because of their uniqueness and consistency over time, fingerprints have been used for identification for over a century, more recently becoming automated due to advancements in computing capabilities. Fingerprint identification is popular because of the inherent ease in acquisition, the numerous sources (ten fingers), available for collection, and their established use and collections by law enforcement and immigration. A fingerprint appears as a series of dark lines that represent the high, peaking portion of the friction ridge skin, while the valleys between these ridges appears as white space and are the low, shallow portions of the friction ridge skin. Fingerprint identification is based primarily on the minutiae, or the location and direction of the ridge endings and splits along a ridge path. Figures 1.1 and 1.2 below present examples of fingerprint features: two types of minutiae and examples of other detailed characteristics sometimes used during the automatic classification and minutiae extraction processes.
In over one hundred forty years of comparison worldwide, no two fingerprints have ever been found to be alike, not even in those of identical twins [Technolvogy.com 2006]. However, experiments performed over time have revealed that fingerprint scanning could be easily tricked. One such method revealed that by simply breathing upon traces of fat left by fingerprints on the scanner’s surface revealed contours of the old fingerprint on the protected PC and granted access [Outlaw.com, 2002]. Also, the gathering of fingerprints is associated with criminal behavior in the minds people and is rejected by many.

1.3.2 Iris Recognition

Iris recognition is the process of recognizing a person by analyzing the random pattern of the iris as shown in Figure 1.3. The automated method of iris recognition is relatively young, existing in patent only since 1994.
The iris is a muscle within the eye that regulates the size of the pupil, controlling the amount of light that enters the eye. It is the colored portion of the eye with coloring based on the amount of melatonin pigment within the muscle as shown in Figure 1.4.

![Figure 1.3 Iris Diagram.](image) ![Figure 1.4 Iris Structure.](image)

Although the coloration and structure of the iris is genetically linked, the details of the patterns are not. The iris develops during prenatal growth through a process of tight forming and folding of the tissue membrane [Hill, 2003]. Prior to birth, degeneration occurs, resulting in the pupil opening and the random, unique patterns of the iris [Westmoreland 1998]. Although genetically identical, an individual’s iris are unique and structurally distinct, which allows for it to be used for recognition purposes.

Such a complex system as the iris has also been proven to fail under experimentation. One such experiment authenticated an unauthorized person into the system by simply holding an ink-jet print-out over their eye. The page was a printout of an authentic iris
with a small hole cut into the page through which the pupil of the imposter was visible to the camera [Outlaw.com, 2002].

1.3.3 Facial Recognition

People often use faces to recognize individuals and advancements in computing capability over the past few decades now enable similar recognitions automatically. Early face recognition algorithms used simple geometric models, but the recognition process has now matured into a science of sophisticated mathematical representations and matching processes. Major advancements and initiatives have propelled face recognition technology into the spotlight. Face recognition can be used for both verification and identification. There are two predominant approaches to the face recognition problem: geometric (feature based) and photometric (view based). As researcher interest in face recognition continued, many different algorithms were developed, three of which have been well studied in face recognition research: Principal Components Analysis (PCA), Linear Discriminant Analysis (LDA), and Elastic Bunch Graph Matching (EBGM).

PCA, Principal Components Analysis commonly referred to as the use of eigenfaces, is the technique pioneered by Kirby and Sirivich in 1988. With PCA, the probe and gallery images must be the same size and must first be normalized to line up the eyes and mouth of the subjects within the images see Figure 1.5. The PCA approach is then used to reduce the dimension of the data by means of data compression basics [Sirovich 1987] and reveals the most effective low dimensional structure of facial patterns. This reduction in dimensions removes information that is not useful [Bolme 2003] and precisely decomposes the face structure into orthogonal, uncorrelated components known as eigenfaces. Each face image may be represented as a weighted
sum of the eigenfaces, which are stored in a one-dimensional array. A probe image is compared against a gallery image by measuring the distance between their respective feature vectors. The PCA approach typically requires the full frontal face to be presented each time; otherwise the image results in poor performance [Bolme 2003]. The primary advantage of this technique is that it can reduce the data needed to identify the individual to $1/1000^{th}$ of the data presented.

Figure 1.5 Standard Eigenfaces: Feature vectors are derived using eigenfaces. [MIT Media Laboratory 2002]

LDA, Linear Discriminant Analysis is a statistical approach for classifying samples of unknown classes based on training samples with known classes [Bolme 2003]. This technique aims to maximize between-class (i.e., across users) variance and minimize within-class (i.e., within user) variance. In Figure 1.6 where each block represents a class, there are large variances between classes, but little variance within classes. When dealing with high dimensional face data, this technique faces the small
sample size problem that arises where there are a small number of available training
samples compared to the dimensionality of the sample space [Lu 2003].

![Image](image.png)

Figure 1.6: Example of Six Classes Using LDA [Lu 2002].

EBGM, Elastic Bunch Graph Matching relies on the concept that real face images
have many non-linear characteristics that are not addressed by the linear analysis methods
discussed earlier, such as variations in illumination, pose and expression. A Gabor
wavelet transform creates a dynamic link architecture that projects the face onto an
elastic grid [Bolme 2003]. The Gabor jet is a node on the elastic grid, notated by circles
on the image below, which describes the image behavior around a given pixel. It is the
result of a convolution of the image with a Gabor filter, which is used to detect shapes
and to extract features using image processing. Recognition is based on the similarity of
the Gabor filter response at each Gabor node [Bolme 2003]. This biologically-based
method using Gabor filters is a process executed in the visual cortex of higher mammals.
The difficulty with this method is the requirement of accurate landmark localization,
which can sometimes be achieved by combining PCA and LDA methods.
Like fingerprint scanning and iris recognition previously mentioned, the three facial recognition methods mentioned below have also been proven to have some flaws under experimentation. Some quotes by airport officials after a facial recognition system was in place were: [ACLU 2002]

- The subject could not be wearing glasses: "Eyeglasses were problematic," according to a summary of the test findings. "Glare from ambient light and tinted lenses diminished the system's effectiveness."

- The angle of the facial image could not vary: "There was a substantial loss in matching if test subject had a pose 15 to 30 degrees (up/down, right/left) off of input camera focal point."

- The subject had to be perfectly still: "Motion of test subject head has a significant effect on the system ability to both capture and alarm on test subject."

- The subject had to be properly lit: "System required approximately 250 lux of directional lighting" to work.
• The airport had to have high quality photographs. "Input photographs populating the database need to be of a good quality."

1.4 Benefits of Biometrics

The goal of any access control system is to authenticate authorized people, not just their credentials, into specific places or allow them to perform specific tasks. Only with the use of a biometric device can this goal be achieved. A card-based access system will control the access of authorized pieces of plastic, but not who is in possession of the card. Systems using PINs require an individual only know a specific number to gain entry; but who actually entered the code cannot be determined.

As opposed to using badges, sign-ins or other ways of tracking employees, a biometric time clock assures that no employee can punch in for another, eliminating time fraud and reducing payroll costs. Because every person’s biometric characteristic is unique, a biometric time clock provides a quick, accurate, and reliable way to record in- and out-punches for each employee. That’s why so many companies now employ biometrics.

1.5 Issues in Biometrics

Issues in biometrics range from cultural, social, and religious issues depending on where the biometric technology is being deployed.

1.5.1 Hygiene

Objections to biometrics based on concerns of cleanliness are one of the major issues concerning scanners. Much as with concerns of the cleanliness of public restrooms,
participants may feel uncomfortable placing their faces against a machine to have their retinas scanned after many others have done so or touching a hand-geometry scanner during flu and cold season.

1.5.2 Religious Objections

Religious objections to biometrics might arise from a variety of different groups. For example, Christians might interpret biometrics to be a “Mark of the Beast.” The objection could be based on the scripture in “Revelation”: The Beast causeth all, both small and great, rich and poor, free and bond, to receive a mark in there right hand or in their foreheads...And that no man might buy or sell, save that he had the mark, or the name of the beast, or the number of his name. . . . and his number is six hundred, threescore, and six [John the Apostle, 96 CE]. Certain Christians could consider the biometric to be the brand discussed in Revelation and biometric readers as the only means of viewing these brands.

1.5.3 Misuse of Data

Misuse of personal information, including the stealing of identities has become more of a threat. Used in certain ways, biometrics provides greater security because the biometric identifier is much harder to steal or counterfeit. On the other hand, where biometrics are authenticated remotely, that is, by transmission of data from a sensor to a centralized data repository, a hacker might be able to steal, copy, or reverse-engineer the biometric. This misappropriation could also come about through insider misuse. Without proper safeguards, files could be misappropriated and transactions could be performed using other people’s identities.
1.6 Vein Map Authentication

On the contrary to biometric authentication technology used today, a system which is based on the vein pattern in the palm of a human hand is being developed. The system is essentially a sensor, much like a fingerprint scanner where the user is required to hold their finger over the scanner. However, with this newly developed vein map authentication technology, the user never actually makes contact with the scanner. The sensor can only recognize the vein map if blood is actively flowing the individual’s veins. Processing is not affected by race, skin discoloration, hair, age, or time. As veins are internal to the body it is extremely difficult to forge the vein pattern of someone else to gain authentication into the system, thereby enabling a very high level of security. The potential market range is very large due to low cost, high accuracy, response timing, size, and the hygienic and friendly use of the system. The device’s way of working will gain public acceptance in the public eye due its contactless and non-intrusive technology.
2. BIOMETRICS AND VEIN MAP AUTHENTICATION

The research in this project introduces vein map authentication and tries to persuade the audience of its significance in the biometric world and highlight key points that give vein map authentication an edge over other biometric systems on the market. The author accomplishes this by implementing a rich user interface using the Visual Basic 2005 programming language interacting with Fujitsu’s C++ library to control a vein map scanner prototype. The author also introduces the strong capabilities of Visual Studio’s CLR’s (Common Language Runtime) ability to interact with unmanaged code (code or dlls written outside of the Visual Studio environment). The scope of the project focuses more on the ability of Fujitsu’s scanner and API (Application Programming Interface) to extract the vein map pattern from various individuals’ palms and the retrieval of the vein map data from a data repository for authentication, as opposed to analyzing the algorithms used for extraction and identification in the C++ library. As shown in the Figures 2.1, 2.2 and 2.3 below, the user interface consists of a tab control with several tabs containing: group boxes, picture boxes, textboxes, and buttons and labels, each displaying a different function of the software system. Section 2.2-2.4 describes the application prototype that the user interacts with.

2.1 Registration Tab

The first tab as displayed in Figure 2.1 is used for registration of the vein map data. The registration process is necessary to introduce the vein data into the vein map system for comparison during the authentication process.
2.2 Authentication Tab

The second tab as displayed in Figure 2.2 is used for authentication. This process extracts the vein map of the individual whose palm is over the scanner and compares the extracted vein map to registered data in the vein map repository.
2.3 Door Control Tab

The third tab as displayed in Figure 2.3 is used as a control system which will authenticate a subject and either unlock a door latch or sound a failure signal. This function of the application is identical to the authentication tab page; however, if the person’s vein map is found in the data repository the PC will apply voltage to a USB digital I/O device provided by SeaLevel Corporation (SEAIO). A separate function in the application will monitor the I/O device for changing occurrences in voltage and send current through a terminal strip to an electronic door latch, thus locking and unlocking the door.
Figure 2.3 Door Switch Tab Page.
3. PROPOSED SYSTEM DESIGN AND IMPLEMENTATION

3.1 Systems in the Project

The vein map authentication software application requires a number of dynamically linked library files owned by Fujitsu Corporation. These DLLs are called from a Visual Basic 2005 project to control a prototype of the vein map scanner also provided by Fujitsu Corporation. For the purposes of the demonstration in this project, there is no licensing needed to use the scanner provided by Fujitsu Corporation. The project is written using the latest version of Microsoft’s Visual Basic release; therefore the production machine requires Visual Studio 2005 Enterprise Edition to be installed on it. After the vein map pattern is extracted from the palm, the DLLs return an array of bytes un-encrypted. The application then encrypts the returned data using one of the encryption classes within Visual Studio environment. The encrypted data is then stored in a Microsoft SQL Server database. Therefore, SQL Server 2000 or SQL Server 2005 is required to be installed on the production machine. The application also requires a PC camera for a facial snapshot during registration. The camera must be installed with all its drivers for the application to make use of the imaging device. See Figure 3.1 for a diagram of the system architecture. Text marked in red indicates software and hardware provided by Fujitsu.
3.2 Systems Needed for Normal Use

The project’s solution result produces an executable file (.exe) that controls and calls the Fujitsu scanner and DLLs. For any other machine to run the application besides the production machine, the following are required: Microsoft’s .Net Framework 2.0 redistributable package, all Fujitsu DLLs installed and registered within the PC registry, the Fujitsu scanner, an imaging device such as a PC camera, and depending on if the application will be run in standalone or client/server mode, the machine also needs to have a version of SQL Server installed for the storage of the vein map data.
3.3 Database tables and Schema

Since the application is to demonstrate how Visual Studio 2005 interacts with the Fujitsu API, the application only requires one table which belongs to the database named VeinMap. The name of the table is also VeinMap and its columns are described below:

- **RegistrationId** – The RegistrationId acts as the unique identifier for each row that is entered into the database. Its data type is nvarchar(100) and no nulls will be accepted as entry for the column. The actual value is not generated by SQL Server, rather it is a Global Unique Identifier (GUID) generated within the application using Visual Studio’s System.Guid.NewGuid class. The GUID is then MD5 hashed and is used as a private key to encrypt the rest of the fields using Visual Studio’s System.Security.Cryptography class.

- **LastName** – The LastName column holds the person’s last name during the registration process. Its data type is nvarchar(100) and also allows no null values to be entered into the column.

- **FirstName** – The FirstName column holds the person’s first name during the registration process. Its data type is nvarchar(100) and also allows no null values to be entered into the column.

- **Hand** – The Hand column holds the person’s left or right hand scanned during the registration process. Its data type is nvarchar(100) and no null values are accepted.

- **TimeOfRegistration** – The TimeOfRegistration column captures the exact time the person’s vein map is registered. Its data type is nvarchar(100) and
stores the date and time in mm/dd/yyyy hh:mm:ss am/pm format. The application automatically generates the current date time stamp and is a part of the SQL UPDATE statement when data is being added to the database. No nulls will be accepted.

- SSN – The SSN column holds the individuals Social Security number that is being registered. Its data type is nvarchar(100) and does not allow nulls to be saved to the database.

- Sex – The Sex column holds the individuals sex (male/female) that is being registered. Its data type is nvarchar(100) and does not allow nulls to be saved to the database.

- Race - The Race column holds the individuals ethnicity that is being registered. Its data type is nvarchar(100) and does not allow nulls to be saved to the database.

- BirthDate - The BirthDate column captures the person's date of birth whose vein map is registered. Its data type is nvarchar(100) and stores the date and time depending on how the user entered it. This column in the database does not allow null values to be entered into the database.

- PalmData – The PalmData column houses the actual encrypted vein map data. Its data type is VARBINARY(MAX) which is new data type in SQL Server 2005. Before, large objects were conventionally stored as binary large object (BLOB) data types. No nulls are accepted.

- Photo – The Photo column contains the facial snap-shot of the individual whose vein map is registered. Its data type is image. Image is also a new
data type in SQL Server 2005. Images were also conventionally stored as
BLOB data types in previous versions on SQL Server. No nulls are
accepted.

Figure 3.2 Sample Vein Map Table in SQL Server 2005.

3.4 Classes and Methods Defined

3.4.1 FVM Class

The FVM Class is the heart and soul of the application. It is essentially the header
file that wraps Fujitsu’s C++ API into the .Net Framework. The class marshals all
memory needed to communicate with the DLLs provided by Fujitsu Corporation. It is
the class where all function definitions are defined along with all the structures and
enumerations that are used to communicate with the vein map scanner. The list of
functions included in the FVM class include:

- **Basic Functions** – Functions used to open/close the library and
  controlling the transaction.

  **Fvm_Init Function**: Opens and initializes vein authentication library

  **Fvm_Term Function**: Closes vein authentication library and frees any
  associated internal resources

  **Fvm_BeginTransaction**: Begins the transaction of registration or
verification process

Fvm_EndTransaction: Ends the transaction of registration or verification process

- **Sequence Functions** – Functions that create registration templates, verify vein data, cancel the execution, etc.

  Fvm_CreateTemplateSequence: Creates vein data for verification for registering database.

  Fvm_VerifyMatchSequence: Verifies vein data for 1:1 authentication.

  Fvm_VerifyMatchMultiSequence: Verifies vein data for 1: N authentication.

  Fvm_CancelSequence: Cancels the processing of

  Fvm_VerifyMatchSequence function and

  Fvm_CreateTemplateSequence function.

  Fvm_Sense Function: Monitors whether palms are located over sensors. Returns when a palm detected.

  Fvm_CreateVeinDataSequence: Creates vein data by scanning a palm. The vein data will be used by function

  Fvm_VerifyMatchVeinData.

  Fvm_VerifyMatchVeinData: Verifies vein data for 1: N authentication.
• **Callback Functions** - Functions that request the user to navigate their hand.

*Message Callback:* Notifies the location where a hand should be placed and operating status of authentication library.

*LpfnSetTemplate* - In the process of *Fvm_VeryfMatchSequence* this Callback Function, which is set to the parameter of *Fvm_VeryfMatchSequence*, will be called by the authentication library for verification data.

*LpfnSetTemplateList:* In the process of *Fvm_VeryfMatchMultiSequence*, this Callback function, which is set to the parameter of *Fvm_VeryfMatchMultiSequence*, will be called by authentication library for verification data.

### 3.4.2 iCam Class

The iCam class is used to control any imaging device installed on a system. Its main function in this application is to capture a snapshot (frame) of the screen given a particular x, y coordinate in pixels. The class uses, user32.dll, GDI32.dll, and avicap32.dll which are shipped and installed with Windows XP. The list of functions included in the class are:

- *System32 Wrapped Functions* – These functions are explicitly defined to interact with the system32 dlls shipped with Widows XP.
SendMessage - Sends messages to the operating system requesting the driver installed, setting the scale of the frame, and other initialization or disposal data.

CapCreateCaptureWindowA - During camera setup, gives camera position to begin reading from initially.

SetWindowPos - When taking as snapshot of the screen, initializes camera to capture frames given a specified location

- User Defined Function - User defined function that either call or receive data from System32 wrapped functions.

initCam - Checks if camera is running, if so, do nothing, else call CapCreateCaptureWindowA

setCam - Sets up camera to take snapshot by calling SendMessage, then SetWindowPos

closeCam - Disposes of camera object by calling SendMessage with disconnect as argument.

3.4.3 VMF Class

The VMF class is critical in that it allocates memory dynamically based on the number of vein map templates in the data repository. The VMF class initializes three arrays: KeyArray(), which holds all the registration keys in the data repository, TemplateArray, which is a two dimensional array that holds in one column, the location of the vein map data in the data repository (acts like a pointer), and in another column the actual vein map data, and PointerArray, which is an array of pointers which points to the pointers
that point to the template data. There is only one method in this class named 
BuildTemplateList. Its purpose is to connect to the veinmap database, 
given a connection string read from an application configuration file, get the 
count of all the templates to compare against, redim (re-initialize) the mentioned 
arrays based on the count received, and load the arrays with the template data.

3.4.4 SEAIO Class

The SEAIO Class interfaces (wraps) the API provided by SEALLEVEL to 
interact with their I/O device which ultimately locks and unlocks a model door 
when a person’s vein map is authenticated. The list of functions the Visual 
Basic application interacts with are: SeaIo_OpenDevice, 
SeaIo_CloseDevice, SeaIo_GetAdapterInfo, 
SeaIo_GetAdapterState, and SeaIo_WriteBit, whose name 
defines their application.

3.5 Information Processing

At application startup, the operator starts by entering the person’s demographic 
information that is registered into the system. This information includes the person’s full 
name, date of birth, social security number, ethnicity, hand, and sex. The application then 
accepts a facial snapshot stored with the demographic information and the vein map data. 

During authentication, the person’s vein map is taken as input into the system, 
after the vein map is extracted and stored in memory, each row in the database containing 
registered vein maps are decrypted and compared until either a match is found or the end 
of the registration data are queried. At that point, the system either presents the person’s 
demographic information along with their facial snapshot if the vein map of the person
being authenticated found a match in the database; otherwise a message appears stating that authentication failed. Figure 3.3 and Figure 3.4 represent the flow of data from the Visual Basic application to the FVM library, to the palm vein scanner, back to the FVM library and finally back to the Visual Basic application.

Figure 3.3 Registration Data Flow.
Figure 3.4 Authentication Data Flow.
3.6 About Fujitsu Vein map Scanner

3.6.1 Specifications

Figure 3.5 below describes the specifications of the Fujitsu prototype that this project uses.

<table>
<thead>
<tr>
<th>Description</th>
<th>PalmSecure Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply voltage (from USB bus power)</td>
<td>4.4 to 5.6V (Input current: up to 500mA)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>2.8W or less</td>
</tr>
<tr>
<td>Interface</td>
<td>USB 2.0 / 1.1 (High speed or full speed)</td>
</tr>
<tr>
<td>Photography distance (from filter surface)</td>
<td>50mm (± 10mm)</td>
</tr>
<tr>
<td>Filter material</td>
<td>Glass</td>
</tr>
<tr>
<td>Acoustic noise</td>
<td>None</td>
</tr>
<tr>
<td>Operating temperature</td>
<td>0°C to 60°C</td>
</tr>
<tr>
<td>Storage temperature</td>
<td>-20°C to 70°C</td>
</tr>
<tr>
<td>Outer dimensions</td>
<td>35mm x 35mm x 27mm</td>
</tr>
<tr>
<td>Weight</td>
<td>50g (approx.)</td>
</tr>
</tbody>
</table>

**Figure 3.5 PalmSecure Specification [Fujitsu 2006].**

3.6 How the Fujitsu Vein Map Scanner Works

The Fujitsu scanner works by capturing a person’s vein pattern image while radiating it with near infrared rays. The deoxidized hemoglobin in the palm vein absorbs these rays, thereby reducing the reflection rate and causing the veins to appear as a black pattern as shown in Figure 3.6. The vein pattern is then verified against a pre-registered pattern to authenticate an individual.

Fujitsu’s proprietary algorithm takes into account identifying features such as the number of veins, their position, and the points in which they cross. Internal research by Fujitsu resulted in a false acceptance rate of less than 0.00007% and a false rejection rate of only 0.00004%. False acceptance rate is a rate at which someone other than the actual person is falsely recognized. False rejection rate is a rate at which the actual person is not recognized accurately [Fujitsu 2006].

28
Figure 3.6 Vein Map after near infrared light Penetration [Fujitsu 2006].
4. TESTING AND EVALUATION

4.1 Application Testing Strategies

Testing was conducted before, during, and after the software in the project was developed. Testing followed the strategic approach as described in [Pressman 2005]. A number of software testing strategies are proposed in the literature. The strategies followed include, technical review by someone other than the author of the project, testing at the component level working outwards, using Microsoft Visual Studio’s rich debugger, and most of all experience and intuition.

4.1.1 Unit Testing

Unit testing focused on the smallest unit of the software design, each component that the application was constructed upon, and the modules that the applications were built from. Each of the four classes that are used in the application where first developed as stand alone programs and rigorously tested before being integrated into the overall product. Below are figures of some of the classes as a standalone application.

![Figure 4.1 SEAIO class as standalone application.](image-url)
Using the component level design description also described in [Pressman 2005] as a guide, the most important control paths were tested using white-box and black-box testing techniques to uncover errors within the boundary of each module. Class FVM.vb itself was not tested at the component level since the class is solely responsible for marshalling all memory and declaring all enumerations and structures that interact with the vein map scanner; its purpose functions as a header file in C++. Its components were tested during integration testing. Each of the components in iCam.vb were tested with its
standalone application, all the components in the class CopyFrame, SetCam, 
SetFrameRate, InitCam, and CloseCam are equally important. Each components has two 
control paths, if..., else..., and were forced to execute at each level. VMA.vb was the 
most critical class to test at the component level since its purpose is to dynamically 
allocate memory at runtime based on the number of vein map templates there are in the 
data repository. Its most critical path can be seen in the following snippet of code:

```vbnet
For i As Integer = 0 To (FVM.SIZEVENBUF - 1)
    If i < myOleDbDataReader.Item(9).Length Then
        TemplateArray(iRow, i) = myVeinMapBytes(i)
    Else
        TemplateArray(iRow, i) = 0
    End If
Next
```

The snippet of code is adding the vein map data to a temporary 2-D array for comparison. 
Like the FVM.vb class the SEAIO.vb class was not tested at the component level in that its 
sole purpose is a wrapper (header) file that has the function definitions to interact with the 
SEALEVEL Corporation API that control their I/O device.

The complexity of each test and the errors that those tests uncovered were limited 
by the constrained scope established for each unit test. Each unit test focused on the 
internal processing logic and the data structures within each component.

### 4.1.2 Integration Testing

Integration testing focused on putting all the modules and components together, 
each class was added and tested to the driving module, frmPalmView.vb, before any 
other was added. After each module was added to the project, instantiation tests were 
conducted to make sure all classes were instantiated and all structures and enumeration 
initialized before use. No problems were uncovered.
4.1.3 System Testing

System testing encompassed a series of different tests whose primary purpose was to fully exercise the program. Testing the application and capturing the Task Manager’s performance report on a Windows XP machine with three quarters of a gigabyte of memory, running on a 3.00GHz, Pentium 4 processor produced the following report seen in Figure 4.6 below.

![Window Task Manager](image)

**Figure 4.4 Windows XP Performance Report during Authentication.**

The same frame was taken of Task Manager’s performance report during authentication on a Windows Vista operating system with 1.1 gigabytes of memory running on a 4.17 GHz, Pentium 4 processor is seen in Figure 4.4.
As can been seen from Task Master’s performance report the software ran more efficiently on the system with a faster processor and more RAM.

These tests ensured that error-handling paths were designed for all information coming from different elements of the system and that bad data or other potential errors in the software were uncovered.

4.1.4 Recovery Testing

Recovery testing ensured that the application handled and recovered from faults in a timely manner. The system was forced to fail in different ways. For example during runtime, a specific device was disconnected intentionally from the workstation causing the application to crash and produce a specific error. Code was added to the application to reinitialize itself before being used again. Also, a specific encrypted column in the data base was manually tampered with to ensure that decryption could not take place if the data were modified outside the application. In this case the application did not recover and that cell in the data repository was corrupt unless changed to its original state.
4.1.5 Security Testing

Security testing ensured that protection mechanisms built into the system protected the application from improper penetration. The system is very secure and robust. All the vein map data previously registered are stored in a SQL Server 2005 database which is password protected to gain access. The data within the repository are symmetrically encrypted using the RijndaelManaged cryptography class provided by Visual Studio, a salt value is read from an application configuratin file, and an MD5 hash of the RegistrationId is used as a private key. Steganography was considered as a mechanism to hide the private key, but opted not to due to image distortion.

4.1.6 Hardware Testing

Each of the hardware devices the application uses were tested independently. The first and most obvious test was to disable the device by removing its respectful driver and starting the application. The first that threw an exception was the iCam class while trying to initialize the camera. The application still ran, however with no imaging device to take a facial snapshot during registration. During registration or authentication, the FVM class then through its own exceptions, stating the palm scanner could not be located. On the contrary, the SEAIO class did not throw an exception when the I/O device driver was disabled. The class simply did not lock and unlock the door during authentication.

The vein map scanner was further tested by scanning a vein map with dirt and oil on the palm. This did not cause the vein map from being properly identified. A zip lock plastic bag was then placed between the palm and the scanner and the identification was not altered. However, placing a cut piece of typing paper between the palm and the scanner caused the scanner to authenticate improperly.
5. CONCLUSION

There are various authentication systems out on the market today, however all have been found to have flaws or are not widely accepted by society. Throughout this research, vein map technology was introduced as a robust authentication system for today’s security need. Proper design and implementation of the proposed authentication system is the cornerstone to its security. Four classes were developed using the Visual Studio programming environment which interact with various hardware devices. The vein map scanner provided by Fujitsu Corporation, a generic pc camera installed with all its drivers and an I/O device provided by Sea Level Corporation. The classes developed in the project interact with the APIs for each of the devices respectfully. Strong emphasis was made during the implementation of the software to security. All data stored and retrieved from the SQL Server 2005 repository were encrypted using the System.Security.Cryptography namespace within the Visual Studio environment.

5.1 Future Work

The software created in conjunction with this project is intended to solely introduce the committee to vein map authentication and programming using the Visual Studio 2005 environment. Continued work using vein map technology should include, research and possible development of the algorithm or algorithms used to extract the vein map from the palm and implementation of sending the authentication data over the internet, e.g., the data repository could be located in Langley, Virginia, while the person under authentication could be in Baghdad, Iraq. There are various USB over IP devices on the market today which could make this possible. Any future work or research to be
conducted using the software developed in this proposal will require the Computer Science Texas A & M University – Corpus Christi to purchase the PalmSecure scanner from Fujitsu Corporation.
BIBLIOGRAPHY AND REFERENCES


[Lu 2002] Lu, J., Boosting Linear Discriminant Analysis for Facial Recognition


APPENDICES

1.1 Source Code

The Source code for each of the four classes developed in the project is listed below.

1.2.1 FVM Class Definition

Imports System.Runtime.InteropServices
Public Class FVM
    Implements IDisposable

    Public Declare Sub Init Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_Init" (ByRef ptOemKey As OEM_KEY, ByRef lptErrInfo As ERRINFO)

    Public Declare Sub Term Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_Term" (ByRef lptErrInfo As ERRINFO)

    Public Declare Sub BeginTransaction Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_BeginTransaction" (ByVal iMode As Integer, <MarshalAs(UnmanagedType.FunctionPtr)> ByVal pCallbackFunc As MessageCallbackDelegate, ByRef pProcessHandle As Integer, ByRef pErrInfo As ERRINFO)

    Public Declare Sub EndTransaction Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_EndTransaction" (ByVal pProcessHandle As Integer, ByRef pErrInfo As ERRINFO)

    Public Declare Sub CreateTemplateSequence Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_CreateTemplateSequence" (ByVal processHandle As Integer, ByVal iCompressMode As Integer, <MarshalAs(UnmanagedType.LPArray, SizeConst:=4096)> ByVal pTemplate As Byte(), ByRef pTemplateSize As Integer, ByRef pErrInfo As ERRINFO)

    Public Declare Sub VerifyMatchSequence Lib "C:\Documents and Settings\All Users\Documents\Visual Studio 2005\Projects\fdpvsb\bin\F3BC1ENG.DLL" Alias "Fvm_VerifyMatchSequence" (ByVal processHandle As Integer, ByVal iNumOfTemplate As Integer, <MarshalAs(UnmanagedType.FunctionPtr)> ByVal pSetTemplateCallback As MatchSequenceCallbackDelegate, <MarshalAs(UnmanagedType.SystemType.SysUInt)> ByVal pResult As IntPtr, ByRef pResult As Integer, ByRef pTemplateNo As Integer, ByRef pErrInfo As ERRINFO)

    Public Declare Sub VerifyMatchMultiSequence Lib "C:\Documents and
Settings\All Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias
"Fvm_VerifyMatchMultiSequence" (ByVal processHandle As Integer,
ByVal iNumOfTemplate As Integer,
<MarshalAs(UnmanagedType.FunctionPtr) > ByVal pSetTemplateCallBack
As MatchMultiSequenceCallbackDelegate, ByVal iVerifyMatchLevel As
Integer, ByVal iCompressMode As Integer, ByVal pResult As Integer,
ByRef pTemplateNo As Integer, <MarshalAs(UnmanagedType.SysUInt)>
ByVal pReserve3 As IntPtr, ByRef pErrInfo As ERRINFO)

Public Declare Sub CancelSequence Lib "C:\Documents and
Settings\All Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias "Fvm_CancelSequence"
(ByVal processHandle As Integer, ByRef pErrInfo As ERRINFO)

Public Declare Sub Sense Lib "C:\Documents and Settings\All
Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias "Fvm_Sense" (ByVal
iTimeout As Integer, ByVal iInterval As Integer, ByVal
iCheckRetryInterval As Integer, ByVal iCheckRetryCount As Integer,
ByRef pErrInfo As ERRINFO)

Public Declare Sub CreateVeinDataSequence Lib "C:\Documents and
Settings\All Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias
"Fvm_CreateVeinDataSequence" (ByVal processHandle As Integer, ByVal
iReserve As Integer, <MarshalAs(UnmanagedType.LPArray,
SizeConst:=4096) > ByVal lpVeinData As Byte(), ByRef pErrInfo As
ERRINFO)

Public Declare Sub VerifyMatchVeinData Lib "C:\Documents and
Settings\All Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias
"Fvm_VerifyMatchVeinData" (ByVal processHandle As Integer, ByVal
iNumOfTemplate As Integer, <MarshalAs(UnmanagedType.FunctionPtr)>
ByVal pSetTemplateCallBack As MatchMultiSequenceCallbackDelegate,
<MarshalAs(UnmanagedType.LPArray, SizeConst:=4096) > ByVal
lpVeinData As Byte(), ByVal iVerifyMatchLevel As Integer, ByVal
iCompressMode As Integer, ByVal pResult As Integer, ByRef
pTemplateNo As Integer, <MarshalAs(UnmanagedType.SysUInt) > ByRef
pReserve3 As IntPtr, ByRef pErrInfo As ERRINFO)

Public Declare Sub GetImage Lib "C:\Documents and Settings\All
Users\Documents\Visual Studio
2005\Projects\fdpvusb\bin\F3BC1ENG.DLL" Alias "Fvm_GetImage" (ByVal
processHandle As Integer, ByVal iKind As Integer,
<MarshalAs(UnmanagedType.LPArray, SizeConst:=309248) > ByVal pvInfo
As Byte(), ByRef piDataSize As Integer, ByRef pErrInfo As ERRINFO)

Public Delegate Sub MessageCallbackDelegate(ByVal iMessageType As
Integer, ByVal iMessage As Integer, ByVal iDataSize As Integer,
ByRef lpvData As MESSAGEINFO)

Public Delegate Function MatchSequenceCallbackDelegate(ByVal
lpNeedNextCallback As Integer, ByVal lpNumOfTemplate As Integer,
<MarshalAs(UnmanagedType.SysUInt) > ByVal lpvTemplate As IntPtr) As
Integer
Public Delegate Function MatchMultiSequenceCallbackDelegate(ByRef lpNeedNextCallback As Integer, ByRef lpNumOfTemplate As Integer, <MarshalAs(UnmanagedType.SysUInt)> ByRef lpvTemplate As IntPtr) As Integer

Private _MessageCallbackAddr As MessageCallbackDelegate
Private _MatchSequenceCallbackAddr As MatchSequenceCallbackDelegate
Private _MatchMultiSequenceCallbackAddr As MatchMultiSequenceCallbackDelegate

Private _OEM_KEY As OEM_KEY
Private _ERRINFO As ERRINFO
Private _FVMpath As String
Private _HANDLE As Integer

' FVM_OEM_KEY Structure
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=24)> _
Public Structure OEM_KEY
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=16)> Dim OemKey As Char()
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> Dim Reserve As Char()
End Structure

' FVM RESOURCEHEADER Structure
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=32)> _
Public Structure RESOURCEHEADER
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> Dim Header1 As Char()
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> Dim Header2 As Char()
    Dim Version As Int16
    Dim Level As Int16
    Dim Length As Integer
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=2)> Dim Reserve As Integer()
    ' Data Area is Below
End Structure

' FVM_ERRORINFO Structure
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=56)> _
Public Structure ERRINFO
    Dim Header As RESOURCEHEADER
    Dim ErrInfo1 As Integer
    Dim ErrInfo2 As Integer
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=4)> Dim ErrInfo3 As Integer()
End Structure

' FVM MESSAGEINFO Structure
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=40)> _
Public Structure MESSAGEINFO
Dim Header As RESOURCEHEADER
Dim iCapturedNow As Int16 ' Inform current scanning counts
Dim iCapturedMax As Int16 ' Necessary total number of scanned data
Dim iReserve As Integer

End Structure

' Enumerated constants used in the program as defined in f3bcleng.h
Public Enum ERCODE
SUCCESS = 0 ' Normal End
CANCELED = 1 ' Cancel
PARAMETER_ERROR = 2 ' Parameter Error
RESOURCE_ERROR = 3 ' Resource Allocate Error
SEQUENCE_ERROR = 4 ' Sequence Error
INTERNAL_ERROR = 5 ' Other Errors
SENSOR_ERROR = 6 ' Sensor Error
TIMEOUT = 7 ' Timeout
SENSE_ERROR = 8

End Enum

' Error Information 1

Public Enum ERR1
SUCCESS = &H0 ' Normal End
CANCELED = &H1 ' Canceled Sequence Function
PARAMETER_ERROR = &H2 ' Parameter Error
SEQUENCE_ERROR = &H3 ' Sequence Error
FUNCTION_ERROR = &H4 ' Function Error
ENGINE_ERROR = &H5 ' Engine Error
SENSOR_ERROR = &H6 ' Sensor Error
CALLBACK = &H7 ' Error in Callback Function
TIMEOUT = &H8 ' Timeout
NO_CANCEL = &H9 ' Can not Cancel
SETTING_ERROR = &HA ' Setting Error
DATA_ERROR = &HB ' Data Error

End Enum

' Error Information 2

' Set Value at Finished Normally
Public Enum SCS
SUCCESS = &H0 ' Normal End
INFORMATION = &H1 ' Information
WARNING = &H2 ' Warning

End Enum

' Set Value at Parameter Error
Public Enum PRMERR
NULL = &H2001 ' NULL is Set
INNER_MEM = &H2002 ' Fault of Internal Memory Area
OUTER_MEM = &H2003 ' Fault of Communication Memory
TEMPLATE_MEM = &H2004 ' Fault of Template Memory Area
VERIFY_MEM = &H2005 ' Fault of Verify Data Area
IMAGE_MEM = &H2006 ' Fault of Image Data Area
MODE = &H2007 ' Fault of Mode Value
HANDLE = &H2008
OEMKEY = &H2009
TIMEOUT = &H200A

INTERVAL = &H200B
CHECK_RETRY_COUNT = &H200C
CHECK_RETRY_INTERVAL = &H200D
COMPRESS_MODE = &H200E

VERIFY_MATCH_LEVEL = &H200F

DATA_KIND = &H2010

End Enum

' Fault of Handle Value
' Fault of OEM Key Value
' Fault of iTimeout Value
(Other than 0/0x7FFFFFFF)
' Fault of iInterval Value
(Other than 50 - 1000)
' Fault of iCheckRetryCount Value (Other than 50 - 1000)
' Fault of iCheckRetryInterval Value (Other than 0 - 15)
' Fault of iCompressMode Value
(Other than FVM_COMPRESS_MODE_UNCOMPRESS, FVM_COMPRESS_MODE_1KB)
' Fault of iVerifyMatchLevel Value
(Other than FVM_VERIFY_MATCH_LEVEL_NORMAL, FVM_VERIFY_MATCH_LEVEL_HIGHEST, FVM_VERIFY_MATCH_LEVEL_HIGH, FVM_VERIFY_MATCH_LEVEL_LOW, FVM_VERIFY_MATCH_LEVEL_LOWEST)
' Fault of iKind Value (Other than 0)

' Set Value at Sequence Error
Public Enum SEQERR
NO_STARTUP = &H3001
STARTUP = &H3002
DUPLICATE_STARTUP = &H3003

NO_BEGIN = &H3004
BEGIN = &H3005
SEQUENCE = &H3006

NO_END = &H3007

SENSE = &H3008
MODE = &H3009

End Enum

' Not Start (Before Fvm_Init)
' Started (Executing Fvm_Init)
' Double Start (Execute same API)
' Not Start (Before Fvm_BeginTransaction)
' Started (Executing Fvm_BeginTransaction)
' Executing Sequence Function
(Executing Fvm_CreateTemplateSequence, Fvm_VerifyMatchSequence)
' State of Finished Sequence Function
(Finished Fvm_CreateTemplateSequence, Fvm_VerifyMatchSequence)
' Not Execute
' Fvm_EndTransaction)
' Executing Sense Function
(Executing Fvm_Sense)
' Fault Sequence to Mode Value

' Sensor Part Number Error
' SRAM Memory Error
' I2C Flash Error
STATE_NEED_NEXT_IMAGE = 10
STATE_BAD_IMAGE = 11
STATE_DARK = 12
STATE_BRIGHT = 13
STATE_UNABLE_GAIN_ADJUST = 14
STATE_MATCHING = 15

End Enum

Public Enum MODE
  UNPROCESS = 0
  REGIST = 1
  MATCH = 2
End Enum

Public Enum COMPRESS_MODE
  UNCOMPRESS = 0  ' 3Kbyte (default)
  OneKB = 1       ' 1Kbyte
End Enum

Public Enum VERIFY_MATCH_LEVEL
  NORMAL = 0      '  normal (default)
  HIGHEST = 1     '  hard
  HIGH = 2        '  a little hard
  LOW = 3         '  a little easy
  LOWEST = 4      '  easy
End Enum

' Kind of Data

Public Enum DATA
  TEMPLATE = 0    ' Registration Data
  MATCHDATA = 1   ' Verification Data
  IMAGE = 2       ' Palm Image
  CAMERAINFO = 3  ' Camera Adjust Data
  GUIDEMESSAGE = 4 ' Guide Message
End Enum

' Kind of Information

Public Const IMAGE_TYPE_SILHOUETTE = 0   ' Silhouette Image

' Identical Person/Other Person

Public Const SELF As Integer = 1         ' Identical Person
Public Const OTHER As Integer = 0        ' Other Person

' Data Size

Public Const SIZE_VEINBUFFER As Integer = 4096 ' Verification Data Size
Public Const SIZE_PALM_IMAGE As Integer = 309248 ' Image Data Size 320KB\_SThumm.jpg

' For Template Callback
Public Const FVM_TRUE As Integer = 1 ' Continue
Public Const FVM_FALSE As Integer = 0 ' End

Public Sub New(Optional ByVal sOEM_KEY As String = "0KJuup2C4kSkJkJKU")

    ' Make sure the structures are of proper size, particularly the
    ' arrays...
    _OEM_KEY = New OEM_KEY
    ReDim _OEM_KEY.OemKey(15)
    ReDim _OEM_KEY.Reserve(7)
    _OEM_KEY.OemKey = sOEM_KEY ' Fujitsu Supplied Key...

    _ERRINFO = New ERRINFO
    ReDim _ERRINFO.Header.Header1(7)
    ReDim _ERRINFO.Header.Header2(7)
    ReDim _ERRINFO.Header.Reserve(1)
    ReDim _ERRINFO.ErrInfo3(3)
End Sub

Public Sub Dispose() Implements IDisposable.Dispose
    ' Probably need to do something here....
End Sub

Public Property MessageCallbackAddr() As
    <MarshalAs(UnmanagedType.FunctionPtr)> MessageCallbackDelegate
    Get
        Return _MessageCallbackAddr
    End Get
    Set(<MarshalAs(UnmanagedType.FunctionPtr)> ByVal value As
    MessageCallbackDelegate)
        _MessageCallbackAddr = value
    End Set
End Property

Public Property MatchSequenceCallbackAddr() As
    <MarshalAs(UnmanagedType.FunctionPtr)>
    MatchSequenceCallbackDelegate
    Get
        Return _MatchSequenceCallbackAddr
    End Get
    Set(<MarshalAs(UnmanagedType.FunctionPtr)> ByVal value As
    MatchSequenceCallbackDelegate)
        _MatchSequenceCallbackAddr = value
    End Set
End Property

Public Property MatchMultiSequenceCallbackAddr() As
    <MarshalAs(UnmanagedType.FunctionPtr)>
    MatchMultiSequenceCallbackDelegate
    Get
        Return _MatchMultiSequenceCallbackAddr
    End Get
Set(<MarshalAs(UnmanagedType.FunctionPtr)> ByVal value As MatchMultiSequenceCallbackDelegate)  
  _MatchMultiSequenceCallbackAddr = value  
End Set  
End Property  

Public ReadOnly Property OEM_KEY_STRUCT() As OEM_KEY  
Get  
  Return _OEM_KEY  
End Get  
End Property  

Public Property ERRINFO_STRUCT() As ERRINFO  
Get  
  Return _ERRINFO  
End Get  
Set(ByVal value As ERRINFO)  
  _ERRINFO = value  
End Set  
End Property  

Public Property HANDLE() As Integer  
Get  
  Return _HANDLE  
End Get  
Set(ByVal value As Integer)  
  _HANDLE = value  
End Set  
End Property  
End Class  

1.2.2 iCam Class Definition  

Option Explicit On  
'Option Strict On  

Public Class iCam  
#Region "Api/consts"  
  Friend Const WM_CAP As Short = &H400S  
  Private Const WS_CHILD As Integer = &H40000000  
  Private Const WS_VISIBLE As Integer = &H10000000  
  Private Const SWP_NOMOVE As Short = &H2S  
  Private Const SWP_NOCOPY As Short = &H4S  
  Private Const WM_USER As Short = &H400S  
  Private Const WM_CAP_DRIVER_CONNECT As Integer = WM_USER + 10  
  Private Const WM_CAP_DRIVER_DISCONNECT As Integer = WM_USER + 11  
  Private Const WM_CAP_SET_VIDFORMAT As Integer = WM_USER + 45  
  Private Const WM_CAP_SET_PREVIEW As Integer = WM_USER + 50  
  Private Const WM_CAP_SET_PREVIEWRATE As Integer = WM_USER + 52  
  Private Const WM_CAP_SET_FRAME As Long = 1084  
  Private Const WM_CAP_COPY As Long = 1054  
  Private Const WM_CAP_START As Long = WM_USER  
  Private Const WM_CAP_STOP As Long = (WM_CAP_START + 68)  
  Private Const WM_CAP_SEQUENTIAL As Long = (WM_CAP_START + 62)  
  Private Const WM_CAP_SET_SEQUENTIAL_SETUP As Long = (WM_CAP_START +
Private Const WM_CAP_FILE_SET_CAPTURE_FILEA As Long = (WM_CAP_START + 20)
Friend Const HWND_BOTTOM As Short = 1
Friend Const WM_CAP_SET_SCALE As Integer = WM_CAP + 53

Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Integer, ByVal wMsg As Integer, ByVal wParam As Short, ByVal lParam As String) As Integer

Private Declare Function capCreateCaptureWindowA Lib "avicap32.dll" (ByVal lpzWindowName As String, ByVal dwStyle As Integer, ByVal x As Integer, ByVal y As Integer, ByVal nWidth As Integer, ByVal nHeight As Short, ByVal hWndParent As Integer, ByVal nID As Integer) As Integer

Private Declare Function BitBlt Lib "GDI32.DLL" (ByVal hdcDest As IntPtr, ByVal nXDest As Integer, ByVal nYDest As Integer, ByVal nWidth As Integer, ByVal nHeight As Integer, ByVal hdcSrc As IntPtr, ByVal nXSrc As Integer, ByVal nYSrc As Integer, ByVal dwRop As Int32) As Boolean
Declare Function SetWindowPos Lib "user32" (ByVal hWnd As Integer, ByVal hWndInsertAfter As Integer, ByVal x As Integer, ByVal y As Integer, ByVal cx As Integer, ByVal cy As Integer, ByVal wFlags As Integer) As Integer

#Region "Private Subs"

Private iDevice As String
Private hWnd As Integer
Private hWndC As Integer
Public iRunning As Boolean
Private CamFrameRate As Integer = 15
Private OutputHeight As Integer = 240
Private OutputWidth As Integer = 360

Public Sub initCam(ByVal parentH As Integer)
'Gets the handle and initiates camera setup
If Me.iRunning = True Then
    MessageBox.Show("Camera Is Already Running")
    Exit Sub
Else
    hWnd = capCreateCaptureWindowA(iDevice, WS_VISIBLE Or WS_CHILD, 0, 0, OutputWidth, CShort(OutputHeight), parentH, 0)
    If setCam() = False Then
        MessageBox.Show("Error setting up Camera")
    End If
End If
End Sub

Public Function setCam() As Boolean
'Sets all the camera up
If SendMessage(hWnd, WM_CAP_DRIVER_CONNECT, CShort(iDevice), CType(0, String)) = 1 Then
    SendMessage(hWnd, WM_CAP_SET_SCALE, True, 0)
SendMessage(hHwnd, WM_CAP_SET_PREVIEWRATE,
CShort(CameraFrameRate), CType(0, String))
SendMessage(hHwnd, WM_CAP_SET_PREVIEW, 1, CType(0, String))

SetWindowPos(hHwnd, HWND_BOTTOM, 0, 0, OutputWidth,
CShort(OutputHeight), SWP_NOMOVE Or SWP_NOSIZE)
Me.iRunning = True
Return True
Else
Me.iRunning = False
Return False
End If
End Function

Public Function closeCam() As Boolean
' Closes the camera
If Me.iRunning Then
    closeCam = CBool(SendMessage(hHwnd,
WM_CAP_DRIVER_DISCONNECT, 0, CType(0, String)))
    Me.iRunning = False
End If
End Function

'takes snap shot of the current frame
Public Function copyFrame(ByVal src As PictureBox, ByVal rect As RectangleF) As Bitmap
If iRunning Then
    Dim srcPic As Graphics = src.CreateGraphics
    Dim srcBmp As New Bitmap(src.Width, src.Height, srcPic)
    Dim srcMem As Graphics = Graphics.FromImage(srcBmp)
    Dim HDC1 As IntPtr = srcPic.GetHdc
    Dim HDC2 As IntPtr = srcMem.GetHdc

    BitBlt(HDC2, 0, 0, CInt(rect.Width),
        CInt(rect.Height), HDC1, CInt(rect.X), CInt(rect.Y),
        13369376)

    copyFrame = CType(srcBmp.Clone(), Bitmap)

    ' Clean Up
    srcPic.ReleaseHdc(HDC1)
    srcMem.ReleaseHdc(HDC2)
    srcPic.Dispose()
    srcMem.Dispose()
Else
    MessageBox.Show("Camera Is Not Running!")
End If
End Function

End Class
1.2.3 VMA Class Definition

Imports System.Data.OleDb

' This class dynamically builds arrays for vein map temporary storage
Public Class VMA
  ' Implements ICollection
  ' Registration Keys
  Private KeyArray(0) As String
    ' 2-Dimensional array to hold veinmap and data row location in the db
  Private TemplateArray(0, FVM.SIZE_VEINBUFFER - 1) As Byte
    ' array of pointers that points to the pointers that point to the template data
  Private PointerArray(0) As IntPtr
  Private _GCHandlePtr As System.Runtime.InteropServices.GCHandle
  Private _StartGCHandle As System.Runtime.InteropServices.GCHandle
  Private _TotalRows As Integer

  Public Function BuildTemplateList() As Boolean
    Try
      ' Selects count of all veinmap templates to compare against
      Dim myOleDbConnection As New
      Data.OleDb.OleDbConnection
      My.Settings.VeinMapConnectionString

      Dim myOleDbCommand As New Data.OleDb.OleDbCommand("Select Count(*) From VeinMap; Select * From VeinMap", _
      myOleDbConnection)
      myOleDbCommand.SelectCommandTimeout = 60
      myOleDbConnection.Open()

      Dim myOleDbDataReader As Data.OleDb.OleDbDataReader = myOleDbCommand.ExecuteReader

      If myOleDbDataReader.HasRows Then
        myOleDbDataReader.Read()

        ' total rows
        _TotalRows = myOleDbDataReader.GetInt32(0)

        ' Now ReDim our xArrays...
        ' Array initialization
        ReDim KeyArray(_TotalRows - 1)
        ReDim TemplateArray(_TotalRows - 1, FVM.SIZE_VEINBUFFER - 1)
        ReDim PointerArray(_TotalRows - 1)

        _GCHandlePtr = System.Runtime.InteropServices.GCHandle.Alloc(_
        (PointerArray, _
        System.Runtime.InteropServices.GCHandleType.Pinned)
        _StartGCHandle = System.Runtime.InteropServices.GCHandle.Alloc(_
        (TemplateArray, _

      _GCHandlePtr = System.Runtime.InteropServices.GCHandle.Alloc(_
      (PointerArray, _
      System.Runtime.InteropServices.GCHandleType.Pinned)
      _StartGCHandle = System.Runtime.InteropServices.GCHandle.Alloc(_
      (TemplateArray, _

      Dim i As Integer = 0
      Do While i < _TotalRows
        KeyArray(i) = myOleDbDataReader.GetString(0)
        TemplateArray(i, 1) = myOleDbDataReader.GetByte(1)
        TemplateArray(i, 0) = myOleDbDataReader.GetByte(0)
        PointerArray(i) = New System.IntPtr(myOleDbDataReader.ParameterCount)
        myOleDbDataReader.Read
        i = i + 1
      Loop
    Catch ex As Exception
      MsgBox("Error building template list: " & ex.Message)
    End Try
  End Function
End Class
myOleDbDataReader.NextResult()
If myOleDbDataReader.HasRows AndAlso iTotalRows > 0 Then
    Dim iRow As Integer = 0
    ' Loads the arrays
    While myOleDbDataReader.Read
        ' Load up the xArrays...
        KeyArray(iRow) = myOleDbDataReader.Item("RegistrationId").ToString
    
        Dim myVeinMapBytes = myOleDbDataReader.Item(9).Length - 1
        As Byte myOleDbDataReader.GetBytes(9, 0, myVeinMapBytes, 0, _ myOleDbDataReader.Item(9).Length)
        For i As Integer = 0 To (FVM.SIZE_VEINBUFFER - 1)
            If i < myOleDbDataReader.Item(9).Length Then
                TemplateArray(iRow, i) = myVeinMapBytes(i)
            Else
                TemplateArray(iRow, i) = 0
            End If
        Next
        myVeinMapBytes = Nothing
        ' Loads the pointers
        PointerArray(iRow) = StartGCHandle.AddrOfPinnedObject.ToInt32 + (iRow * FVM.SIZE_VEINBUFFER)
        iRow += 1
    End While
    If Not iTotalRows = iRow Then
        MsgBox("BuildTemplateList Method's rows did not match!" & vbCrLf & iTotalRows.ToString("#,##0") & vbCrLf & iRow.ToString("#,##0"), MsgBoxStyle.Critical)
    End If
End If
myOleDbDataReader.Close()
myOleDbCommand.Dispose()
myOleDbConnection.Close()
myOleDbConnection.Dispose()
BuildTemplateList = True
Catch ex As Exception
    MessageBox.Show(ex.ToString)
    BuildTemplateList = False
End Try
End Function
Public Sub DestroyArrays()
    Try
        ReDim KeyArray(0)
        ReDim TemplateArray(0, 0)
        ReDim PointerArray(0)
        _GCHandlePtr.Free()
        _StartGCHandle.Free()
        _GCHandlePtr = Nothing
        _StartGCHandle = Nothing
    Catch ex As Exception
        MessageBox.Show(ex.ToString)
    End Try
End Sub

Public ReadOnly Property GCHandlePtr() As IntPtr
    Get
        Return _GCHandlePtr.AddrOfPinnedObject
    End Get
End Property

Public ReadOnly Property Length() As Integer
    Get
        Return PointerArray.Length
    End Get
End Property

Public ReadOnly Property Key(ByVal iNumOfTemplate As Integer) As String
    Get
        Return KeyArray(iNumOfTemplate - 1)
    End Get
End Property

Public ReadOnly Property TemplateData(ByVal iNumOfTemplate) As Byte
    Get
        Dim myBuffer(FVM.SIZE_VEINBUFFER - 1) As Byte
        For i As Integer = 0 To FVM.SIZE_VEINBUFFER - 1
            myBuffer(i) = TemplateArray(iNumOfTemplate, i)
        Next
        Return myBuffer
    End Get
End Property

End Class

1.2.4 SEAIO Class Definition

Imports System.Runtime.InteropServices

' Api Return Error Codes
Enum SEAIO_ERROR
    NONE = 0
PARAM = 1
NOT_SUPPORTED = 2
PORT_NOT_OPEN = 3
INVALID_NAME = 4
SHARING_VIOLATION = 5
DRIVER_TYPE = 6
BUFFER = 7
INCOMPLETE = 8
GENERAL = 9
IO_PENDING = 10
NO_TXBUFFER = 11
INVALID_DEVICE_REQUEST = 12
INVALID_HANDLE = 13
CANCELED = 14
LAST = 15
NO_MEMORY = 16
BUSY = 17
INTERRUPT_BUFFER_OVERFLOW = 128
End Enum

'This is the adapter_info structure. It is defined in the SeaIO help
'API definitions.
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi)> _
Public Structure ADAPTER_INFO
'Future Use...
Dim dwReserved0 As Integer
Dim dwReserved1 As Integer
Dim dwReserved2 As Integer
Dim dwReserved3 As Integer

'system Resources
Dim dwStateCapabilities As Integer

'Adapter Setup
Dim wCardNumber As Int16
Dim wBaseIO As Int16

Dim ucPortInCount As Byte
Dim ucPortOutCount As Byte
Dim ucPortCount As Byte

Dim ucEnabled As Byte
Dim wIRQ As Int16 'interrupt for future use
End Structure

Enum AccessModes
Relative = 0
Absolute = 1
End Enum

'This is the adapter_state structure. It is defined in the SeaIO help
'API definitions.
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=37)> _
Public Structure ADAPTER_STATE
Dim dwInfoBits As Integer
Dim dwOutputPreset As Integer
Dim dwInterruptBufferSize As Integer
Dim dwInterruptControlWord As Integer
Dim dwSampleInterval As Integer

'Array that holds setup info for board with software configurable
'input / output capability. The format of words are defined in
'each individual manual. The format for the 4030 is defined in the
'8255 manual.
<MarshalAs(UnmanagedType.ByValArray, SizeConst:=9)> Dim ModeCW As
Byte()
<MarshalAs(UnmanagedType.ByValArray, SizeConst:=8)> Dim AIRange As
Byte()
End Structure

'This is the overlapped structure. It is defined in the WinBase help
'API definitions.
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi, Size:=20)>

Public Structure OVERLAPPED
  Dim Internal As UInt64
  Dim Offset As Int64
  Dim hEvent As IntPtr
End Structure

Public Class SEAI0
  Public hObject As Integer
  Public PortNum As Integer
  Public ReturnCode As Integer
  Public NotifyInputChangeThread As Threading.Thread
  Public AdapterInfo As ADAPTER_INFO
  Public AdapterState As ADAPTER_STATE

  'SeaI032 Functions
  Declare Function SeaI0_OpenDevice Lib "seaio32.dll" (ByVal wNum As
  Byte, ByRef pHandle As Integer) As Integer

  Declare Function SeaI0_CloseDevice Lib "seaio32.dll" (ByVal HANDLE
  As Integer) As Integer

  Declare Function SeaI0_GetAdapterInfo Lib "seaio32.dll" (ByVal
  Handle As Integer, ByRef pStruct As ADAPTER_INFO) As Integer

  Declare Function SeaI0_GetAdapterState Lib "seaio32.dll" (ByVal
  Handle As Integer, ByRef pStruct As ADAPTER_STATE) As Integer

  Declare Function SeaI0_WriteBit Lib "seaio32.dll" (ByVal HANDLE
  As Integer, ByVal lngBitNumber As Integer, ByVal boolState As Boolean,
  ByVal intMode As Int16) As Integer
End Class

1.2 Copy Rights

CBM Archives Co. has intellectual property rights relating to the technology
embodied in this document. In particular, and without limitation, these intellectual
property rights may include one or more U.S. patents or pending patent applications in
the U.S. and/or other countries. The software described in this document were developed
under licenses restricting its use, copying distribution, and recompilation. No part of this
software may be reproduced in any form by any means without prior written
authorization of CBM Archives Co.

Printed in the United States of America. Revision: 04/31/07 4:01 PM

© Copyright 2007 CBM Archives Co. All rights reserved.